Construction and Analysis of Neural Radiance Fields (NeRF)

Quinn Arbolante **Peter Li**

Abstract

We attempt to synthesize novel views of complex static scenes using Neural Radiance Fields (NeRF) [4]. Furthermore, we attempt to replicate and expand on previous work done with fully-connected deep neural networks to improve the quality of NeRF generated images. We describe our process of recreating the standard NeRF model, our experiments, and finally our attempted modifications to the positional encoder. Code can be found in our Github repository.

1 Introduction

Novel view synthesis is a popular problem in computer vision: given input images of a scene and their respective camera poses, can we create images of the scene at different camera poses? This problem is significant as it is closely related to other computer vision problems such as 3D scene reconstruction and even video frame interpolation. It is also very applicable to solve common problems such as video stabilization.

A recent groundbreaking method of novel view synthesis uses Neural Radiance Fields to represent scenes for view synthesis. This algorithm takes in a spatial locations (x, y, z) and viewing directions (θ, ϕ) as input and outputs the color and opacity (likelihood the point is occluded) via a deep neural network. Once this function is trained for a specific (static) scene, it can be used to create an image of the scene from a camera pose via volume rendering, an alternative rendering method from the more popular 3D rendering technique in computer graphics.

NeRF has been built heavily upon and extended, from being used on dynamic scenes, to removing the need for camera pose information from input images, to being made faster. We reproduce NeRF, utilizing Matthew McGough's article and the original NeRF code as a guideline. We train and test on data from the original NeRF paper.

2 Background and Related Work

There have been numerous extensions, modifications, and adjustments made to NeRF since it's public release in 2020 [4]. Some works adapt NeRF to space-time synthesis on dynamic scenes [3], novel view synthesis of an object in different environments at different points in time [2], and even image synthesis on multiple objects in a scene [5]. Each of these works expands on NeRF using a new novel technique, whether by increasing the inputs and outputs and modifying the architecture, applying it in a different problem setting as one of the model's many parts, and composing multiple NeRFs to separate data.

There exists several works focused on improving NeRF or expanding its versatility. NeRF++ [9] does a deep technical analysis of NeRF and observes its strengths and weaknesses. The work a discovers multiple limitations with the NeRF model-when presented with 360 degree videos NeRF

Submitted to 34th Conference on Neural Information Processing Systems (NeurIPS 2020). Do not distribute.

severely under performs-and adresses these issues with novel solutions. Google Research created urban radiance fields (URF) [6], which adds lidar data as input to the model and performs 3D scene reconstruction. One of the most interesting papers perhaps is Plenoxels [8], a competitor to NeRF, which models a radiance field as a sparse grid of points, but does not require a neural network to train. This feature enables and the model to train in significantly less time.

Overall, there has been plenty of work exploring NeRF. In this paper, we look at reconstructing the original NeRF.

3 Methods

We start with the traditional NeRF architecture. For the neural radiance field, NeRF uses a simple deep neural network with 9 fully-connected layers (a multi-layer perceptron).

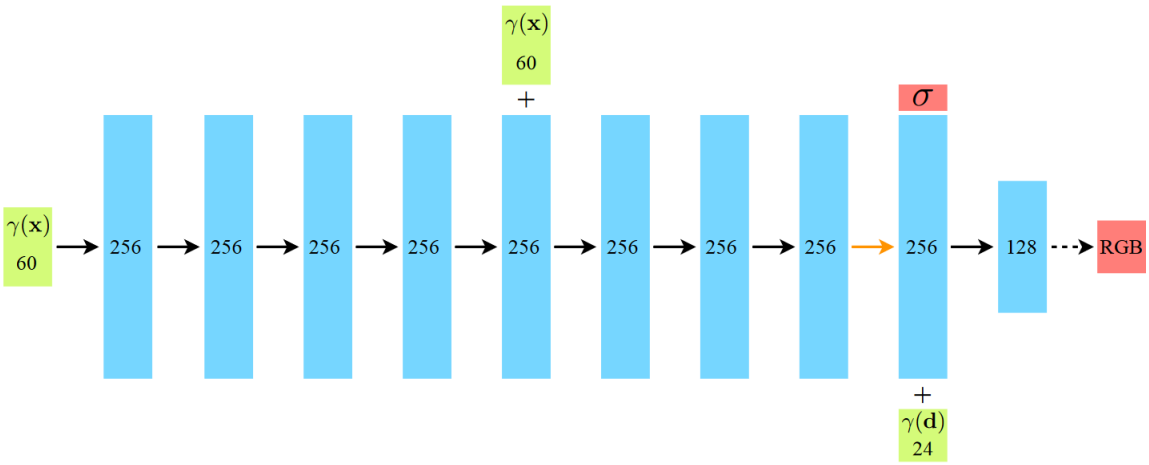

Figure 1: NeRF architecture. [4]

The positional input, $\vec{x}$, goes through a positional encoder γ defined as

$$\gamma(x) = (cos(2^0\pi x), sin(2^0\pi x), ..., cos(2^L\pi x), sin(2^L\pi x)$$

Where L is a hyper-parameter. A residual connection is added to the fifth layer, and the viewing direction is only added at the ninth layer. The viewing direction is added near the end so that the model focuses on the position to determine the output color and opacity, since that should be the primary factor. Opacity, denoted σ, comes from the ninth layer, whereas color goes through an additional layer before being returned.

The loss function uses volume rendering to recreate images that can be compared to test images. Volume rendering works as follows: given a camera pose, a near and a far plane, and the image width and height, rays are sent from the camera to each pixel. For each ray, points are randomly sampled between the near and far plane, and their weighted sum results in the final color of the pixel of the recreated image. The opacity at each point is used as the weights. This technique is known as ray marching.

Additionally, when computing the expected color, an additional function $T(t)$ is used to represent accumulated transparency; the farther away something is, the less the eye can see it. Accumulated transparency is represented with the following function:

$$T(t) = -\int_{t_n}^{t} \sigma(\vec{r}(s))ds$$

Note that t is the parameterization of the ray, with t_n and t_f being the values for the near and far plane intersecting the ray. The final equation for the expected color of a pixel, denoted $\hat{C}$, is

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

Once we've recreated the image, we can take the mean squared error between $\hat{C}$ and the ground truth C as our loss.

The original paper makes an optimization. Since the 3D space of the scene is usually sparsely filled with interesting points, they create two networks: one where points are sampled along the ray approximately evenly spaced (stratified sampling) and one where points are bias to be closer together, ideally around interesting areas of the ray. This forms a coarse network and a fine network, where the fine network learns from the coarse network where to better sample. The original paper calls this hierarchical volume sampling.

We first try to reconstruct the smallest version of NeRF, known as Tiny-NeRF. Tiny-NeRF is a Tensorflow implementation of a simplified NeRF model; we make a PyTorch implementation. Then, we try to recreate the full NeRF model.

Due to issues with training for long periods of time, we instead sample twice in order to get better points to compute $\hat{C}$ with; once with stratified sampling, then a second time by looking at the initial samples and determining the best areas to sample from.

4 Experiments

Here is Tiny-NeRF's output and our output after 500 training iterations. Note that this is a very small number of iterations relative to the original model.

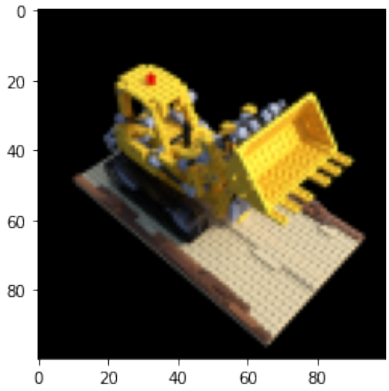

Figure 2: Original image

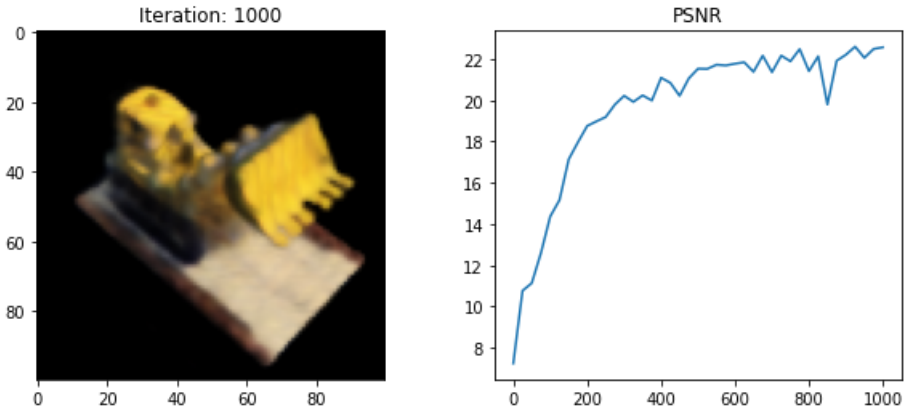

Figure 3: Tiny-NeRF output

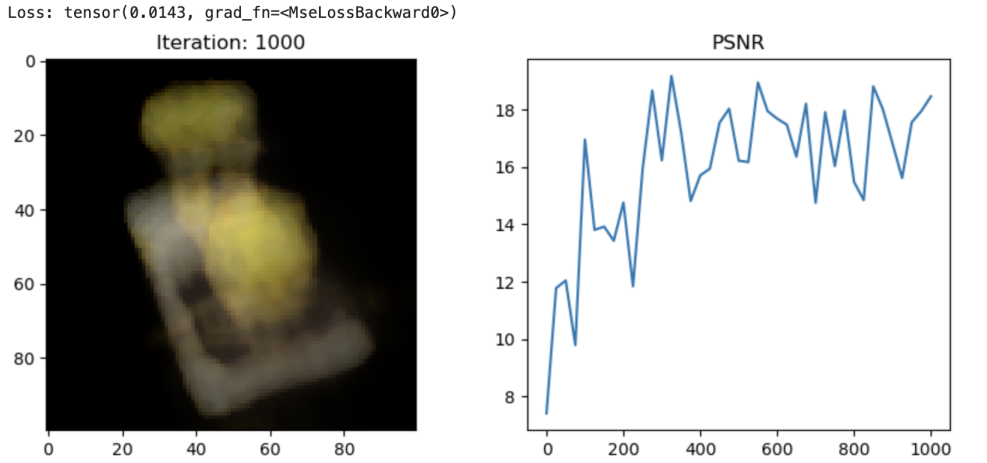

Figure 4: Our output

We also have the peak signal-to-noise ratio (PSNR).

It is clear that the original NeRF implementation works better. However, with different numbers of iterations in training the models, it is difficult to compare. One key similarity between both models is the spikiness in the peak signal-to-noise ratio; this is a result of the positional encoder and how neural networks approximate. Further research into this topic resulted in us finding a paper from the original authors of NeRF on how neural networks are biased towards learning low-frequency functions [7] and how to counteract this by changing the positional encoder. We tried to implement the random Fourier features encoder mentioned in the paper, but were unable to do so.

Though the models were not trained enough, you can still recognize the Lego in the image. However, the image is quite blurred depending on the model. This is also suspected to be a result of the positional encoding; in another followup paper by the authors [1] they show that adjustments to the hyper-parameters of the random Fourier feature encoding can improve the output.

5 Conclusion

We tried to reconstruct NeRF based off of the original paper. Though we failed in various ways (inability to train model long enough, simplifications to paper), we still found it a great learning experience and learned a lot about training large deep neural networks, radiance fields, and volume rendering. For the both of us, this project taught us a lot about training models, loss functions, reading research papers, and even high performance computing.

There are many different approaches and further work that can be done. Though there does exist good NeRF implementations in various machine learning frameworks (Tensorflow, PyTorch), there does not exist alternative implementations for some of the works that use NeRF. Our initial goal was to create implementations for these works (e.g. Neural Scene Flow Fields), and though we were unable to reach that goal, it still remains. There are plenty of optimizations that can be done to our existing reconstruction (better positional encoding, coarse and fine network), so that is also a straightforward future step.

References

[1] Jonathan T. Barron, Ben Mildenhall, Matthew Tancik, Peter Hedman, Ricardo Martin-Brualla, and Pratul P. Srinivasan. Mip-nerf: A multiscale representation for anti-aliasing neural radiance fields. *ICCV*, 2021.

[2] Zhengfei Kuang, Kyle Olszewski, Menglei Chai, Zeng Huang, Panos Achlioptas, and Sergey Tulyakov. NeROIC: Neural object capture and rendering from online image collections. *Computing Research Repository (CoRR)*, abs/2201.02533, 2022.

[3] Zhengqi Li, Simon Niklaus, Noah Snavely, and Oliver Wang. Neural scene flow fields for space-time view synthesis of dynamic scenes. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2021.

[4] Ben Mildenhall, Pratul P. Srinivasan, Matthew Tancik, Jonathan T. Barron, Ravi Ramamoorthi, and Ren Ng. Nerf: Representing scenes as neural radiance fields for view synthesis. In *ECCV*, 2020.

[5] Michael Niemeyer and Andreas Geiger. GIRAFFE: representing scenes as compositional generative neural feature fields. *CoRR*, abs/2011.12100, 2020.

[6] Konstantinos Rematas, Andrew Liu, Pratul P. Srinivasan, Jonathan T. Barron, Andrea Tagliasacchi, Tom Funkhouser, and Vittorio Ferrari. Urban radiance fields. *CVPR*, 2022.

[7] Matthew Tancik, Pratul P. Srinivasan, Ben Mildenhall, Sara Fridovich-Keil, Nithin Raghavan, Utkarsh Singhal, Ravi Ramamoorthi, Jonathan T. Barron, and Ren Ng. Fourier features let networks learn high frequency functions in low dimensional domains. *NeurIPS*, 2020.

[8] Alex Yu, Sara Fridovich-Keil, Matthew Tancik, Qinhong Chen, Benjamin Recht, and Angjoo Kanazawa. Plenoxels: Radiance fields without neural networks. *CoRR*, abs/2112.05131, 2021.

[9] Kai Zhang, Gernot Riegler, Noah Snavely, and Vladlen Koltun. Nerf++: Analyzing and improving neural radiance fields. *CoRR*, abs/2010.07492, 2020.