Neural Radiance Fields (NeRFs) for Novel View Generation

Philip Sigillito
University of Nebraska Omaha
College of Information Science & Technology
Omaha, USA
psigillito@unomaha.edu

***Abstract*—Neural Networks have been used to encode three dimensional scenes and objects for many years. Similar to other generative models such as GANs, this encoding has allowed the generation of novel views of the scene that are not part of the training set. Neural Radiance Fields, NeRFs, were introduced in 2020 and have since become popular models for encoding these scenes. NeRFs use a Multilayer Perceptron Network to encode the scene as a continuous function in its latent space. Whereas other networks have encoded the scene with voxels, NeRFs aim to learn the radiance field. The radiance field can be thought of as how light travels through the scene. Once the scene is encoded, any position in the scene can be sampled for a specific color and opacity value. This allows an image to be generated for any input camera coordinate and viewing direction. NeRFs also use positional encoding to encourage the MLP to learn high frequency functions and hierarchical sampling to better sample sections of the scene that contribute to pixel color.**

This paper looks at the architecture of NeRFs and how they show such effective results. We then implement an extended version of the tiny_nerf architecture by adding a fine model, hierarchical sampling, and editing the MLP architecture to compare results. We find that the architecture of the MLP does not have a significant impact on the ability for the model to encode the scene. Additionally, we show that positional encoding has a significant effect on the network's ability to encode the scene with fine detail. This benefit is not limited to NeRFs and can be applied to other networks that employ MLPs.

Keywords—Neural Radiance Field, Multilayer Perceptron, Volumetric Rendering

I. Introduction

Generative networks such as GANs are very effective at creating novel 2D images. The logical continuation of generating two dimensional images is to generate 3D scenes. Similar to how GANs train from discrete 2D images, researchers want to train a network to represent an entire 3D scene with only a limited set of discrete 2D images. Traditional techniques for representing 3D space such as voxel grids, meshes, and point clouds have been combined with neural networks to represent scenes by researchers for many years.

These models attempt to represent the 3D scene as a continuous function in the latent space of their model. Once the scene is represented as a continuous function, different points on the function can be queried from the model to display novel views of the scene. Importantly, these views are completely generated by the model and are not included in the training set.

In 2020, [1] introduced Neural Radiance Fields (NeRFs). NeRFs gained in popularity because they provide high detail novel views that outperform previous models. NeRFs also avoided memory expensive representations such as voxel occupancy grids.

NeRFs diverge from previous models in their application of light fields to describe the scene[1]. Light fields describe how the light moves through the scene (i.e. the radiance or reflection off of a surface in the scene). Once the light field has been modeled, any point in the field can be sampled. Once NeRFs learn the light field of a scene they use volumetric rendering techniques to output an image.

NeRFs also employ concepts borrowed from computer graphics to improve performance. The two notable improvements are hierarchical sampling during ray marching and positionally encoding inputs before they are processed. These two techniques are described later in this paper.

Encoding scenes into a model has become the focus of research for many applications. One such field of research is autonomous navigation. Autonomous systems use sensors such as cameras and lidar to measure their environment. As discussed in [4], an autonomous machine that can encode its surroundings can more sophistically interact with its environment because it understands its environment and is not simply reacting to it. Another useful application is in the generation of data sets for research. Once a scene is fully encoded, any view inside the scene can be rendered. This allows for an infinite number of views and positions of a scene to be recorded. Finally, there are many applications in the field of computer graphics. Representing a scene or object with weights for a neural network can be much more compact than a 3D mesh[1]. It can also allow for modeling real world objects much

faster than traditional approaches. For example, scanning real world objects is a common technique but is expensive and time consuming. It also does not perform well for thin materials such as chain link fences, transparent surfaces, or non-solid objects such as smoke[3]. In contrast, a neural representation only requires a small set of discrete images and their camera positions to represent the entire scene.

II. Objectives

NeRFs are relatively new and have created a renewed interest in novel view synthesis. This is especially the case for the computer graphics community. This paper aims to explore the architecture of NeRFs and implement a simple NeRF network to render novel views of an object. Once a simple network has been built, modifications to the network's architecture such as depth, layer size, and activation functions will be applied. These changes will be measured to compare their impact on performance. This paper uses PSNR, peak signal-to-noise ratio, as the quantitative measure of image quality as it appears to be the most commonly accepted metric for image generation quality. Images generated from the models will also be included. Before getting into the implementation, we will first look at some of the relevant literature that has been published on the topic. We will then go through how the NeRF operates, the parameters of our tests and finally the results.

III. Relevant Research

Researchers have been studying the problem of novel view synthesis for years and there is a lot of relevant research. Listed below are several other approaches to novel view synthesis and 3D scene representation.

One common technique is to encode the 3D scene through the use of voxels. Voxels are three dimensional pixels that have a color, opacity, and volume. Voxels can be arranged into a three dimensional grid to represent the scene. Paper [5] proposed a model called VoxNet which utilizes voxels in this manner. Importantly, VoxNet does not encode a scene but uses voxels to encode the object's volume for classification. Its inputs are not images but point cloud data. The voxel occupancy grid is then passed through a convolutional network to classify the object. VoxNet encodes object geometry in its latent space but its reliance on point cloud data is not conducive to novel view synthesis. Although point cloud data is useful for describing volume, it does not provide information regarding color or opacity. For this reason, the use of GRBA (green, red, blue, and alpha for opacity) images as input is superior for generating realistic images.

Deep Voxels is a model proposed in [2] which uses a voxel occupancy grid more robustly than [5]'s object classification. DeepVoxels is designed to be a generative model for 3D feature mapping. As described by its authors, it combines two dimensional and three dimensional techniques to create novel views. Similar to NeRFs, it also uses discrete images of the scene for training. The Deep Voxels' architecture is complicated but at its center it consists of two main networks: one that encodes the three dimensional scene using three dimensional operations and a second network that uses two dimensional operations for rendering fine features[2]. The 3D scene network uses a gated recurrent unit for each voxel in the scene. This causes each voxel to be updated jointly[2]. The 2D Rendering network uses a 2D U-Net architecture.[2] The model also includes an occlusion network between the 3D and 2D networks. The occlusion network reasons about the voxel ordering in relation to the viewer and determines which voxels are visible. NeRFs similarly account for occlusion but are more elegant in their implementation. NeRFs account for occlusion during ray marching. Of the models researched for this report, Deep Voxels is the most complicated.

Neural Volumes, as discussed in [4], use a voxel representation but greatly simplify the rendering process. The neural volume model in [4] uses an encoder and decoder neural network to generate the GRBA volume. Interestingly, [4] also showed that for efficient object representation, it could attain reasonable results with three orthogonal images. It then renders the scene using ray marching with no learned parameters[4]. Ray marching is a technique commonly used in computer graphics to determine a pixel's color. To determine the pixel's color, a ray is cast from the viewer's optical center through the pixel and into the scene. The line from the viewer's optical center through each pixel in the image has a slightly different direction. The ray then marches through the scene until it hits a surface. That surface's color and radiance is sampled to determine the pixel's color.

Traditional ray marching ensures the ray does not pass through a surface. It does this by only advancing each iteration the distance to the closest surface in the scene. Ray marching is sometimes referred to as sphere tracing as the distance of each step can be determined by expanding a sphere from each point until it hits a surface. Notably, [4]'s implementation does not sample the final surface color but samples color and opacity along the ray at regular intervals. It then accumulates the values to handle situations such as transparent and semi-transparent materials.

Perhaps the most unique step [4] implements is that it uses a 'warping' technique to avoid the regular grid structure in its voxel representation. It introduces a warp field which, instead of directly sampling color and opacity along the ray, it samples a 3D warp field encoding of each location along the ray. This simulates a non uniform sampling grid and allows for finer detail than the voxel resolution [4].

Additionally, [4] applies warp fields to adjust the size of voxels that represent empty space and reduce the total number of voxels in the model. This is very beneficial as a common criticism of voxel representation is that it wastes a lot of memory representing empty space.

Generative Query Network, GQNs, are discussed in [3] from a desire for autonomous robotics to understand their surroundings. It suggests two networks: one for neural scene representation and one generative network. The representation network takes in sensor input and creates a neural scene representation. The generative network predicts the scene at an arbitrary location based on the output of the representation network. Interestingly, [3] also compared GQNs to VAE networks with a t_SNE two-dimensional t-distributed stochastic neighbor embedding of scene representations. It showed that images from the same scenes were grouped together for the GQN network while the VAE network's images showed no clustering. Paper [3] states that while the GQN clusters were based on which scene they belonged to, the VAE clustered images based on wall angles in the images. This is interesting because according to paper [3] it suggests that the GQN learned the underlying layout of objects whereas the VAE simply learned to represent the objects.

Scene Representation Networks, SRNs, are discussed in paper [4]. SRNs also come from a desire for three dimensional navigation using discrete two dimensional input. This model uses an auto encoder and decoder structure. It uses ray marching similar to [4]'s Neural Volumes and NeRFs. This paper also discusses generalizing weights between similar classes of objects so that the model starts with weights closer to their optimal values. This allows for faster training and the ability to train from a single image. This same idea is applied in presentation [8] for NeRFs.

In [8], it is shown that a NeRF that is initialized with generalized weights can be trained with reasonable accuracy from one image. For example, a network could be given the generalized weights for the furniture class. Then the network could train on a single image of a chair and be able to create reasonably accurate novel views of the chair. Presentation [8] also demonstrates what happens when a NeRF with no preset weights is trained using a single image. Unsurprisingly, the network represents the object almost perfectly in the exact position of the trained image but any shifting of the view results in meaningless noise. This makes sense as the network is overly fit to the single image.

There are other interesting advancements with NeRF models that extend beyond the standard NeRF model introduced in 2020. For example, the authors of [1] introduce the MIP NeRF in paper [11]. This model uses MIP mapping techniques to handle issues of aliasing. NeRFs struggle with rendering areas where there is a resolution change in the image. For example, the boundary between the surface of an object in the foreground and the out of focus background. MIP mapping is a rendering technique where a lower texture resolution is used for sampling objects that are far away.

Another interesting NeRF variant is NeRF-W or NeRF in the Wild as explained in paper [12]. This variation is designed to handle image sets where the scene conditions are not consistent. [12] demonstrates this with the photo tourism data set which includes images of landmarks but the lighting and weather is not consistent between images. There may also be temporary occlusions in the images such as pedestrians. Paper [12] overcomes this issue using generative latent optimization techniques to learn which parts of the image are transient and static. This is important because standard NeRFs such as the one implemented in this paper assumes the scene is static and cannot differentiate transient features in the images.

IV. NeRF Architecture

Neural Radiance Fields combine concepts from deep learning and the computer graphics field. Inputs to the network are 5D coordinates: the camera's x,y,z position as well as theta and phi values which indicate the camera's viewing direction. Once trained, the output of the network is the volume density and emitted radiance (pixel colors) for the 5D coordinates.

NeRFs use a fully connected deep network to encode this continuous function. No convolutions are used in the network; therefore, NeRFs can be considered a multilayer perceptron[1]. Its network optimizes a continuous function in its latent space that represents the scene[1].

NeRF architectures can be any MLP that processes input as a 3D coordinate and viewing direction. In the original NeRF paper, [1], the NeRF consists of two MLP models. The first model finds coarse details. This model's output is then passed to a second model which finds final details and renders the image. Each model in the original paper consists of eight fully connected layers. Each hidden layer uses ReLU for its activation function and all layers have 256 channels per layer.

The network outputs the scene density representation and a 256 dimension feature vector. Halfway through the models is a skip layer connecting the original input. NeRFs use gradient descent to optimize between known images and generated images. One special step NeRFs use when training their models is to preprocess sample ray points with positional encoding.

The network predicts the image pixel by pixel. Although a camera looking direction is given as input by the user, the directions used by the model are the directions from the camera's optical center through each pixel and into the scene. The user's direction is the direction from the camera's optical center through the center point of the image. For this reason, each pixel's direction is slightly off from the user's input. NeRFs use a ray marching technique called stratified sampling discussed in the next section for every pixel ray. These values are what are used by the MLP for encoding the light field.

V. Stratified Sampling and Volume Rendering

NeRFs use traditional volume rendering techniques that rely on volume density to render the image. Volume density is the probability of a ray terminating at a given position. As outlined in [1], the expected color of a pixel is determined by marching a ray through each pixel and into the scene. NeRFs use a technique called stratified sampling which is similar to the ray sampling in [4]. In this technique, the ray is divided into bins between the near and far ends of the scene. A point is then sampled from inside each bin. As outlined in [1] the color of a pixel,$C(r)$, can be formalized as:

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

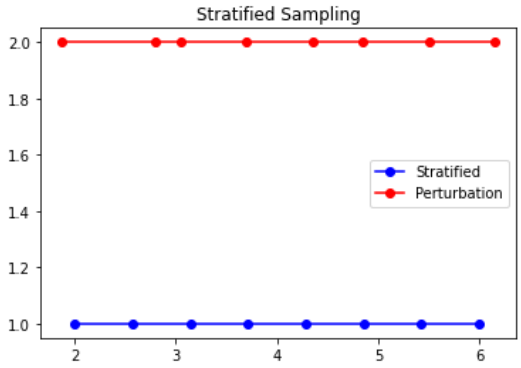
where:

$$T(t) = exp(-\int_{t\,near}^{t} \sigma(r(s))ds)$$

$t\,far$ and $t\,near$ are the bounds of the ray. $T(t)$ is the accumulated transmittance along the ray from near to our point. $\sigma(r(t))$ is the volume density at $r(t)$. $T(t)$ can be thought of as the probability that the ray reaches t without first hitting a surface [1].

The values of the sample's color are accumulated to get the pixel color. Each sample's contribution to the color is weighted by its opacity and order within the ray. If a sample has high opacity, it will diminish the contribution of following samples. As a result, the pixel color is a soft combination of all samples along the ray [1].

In normal stratified sampling, the samples are taken at regular intervals. For NeRFs this would cause a rigid limit to resolution. Perturbation is introduced to approximate continuous sampling. Perturbation causes each sample to be taken in a random position inside its bin. For one ray projection, the sampling is still discrete but over the course of training with hundreds of thousands of ray projections, the many discrete semi-random samples approximate continuous sampling [9]. Depicted below is a sample ray that is stratified (blue) and stratified with perturbation (red). Demonstrating the difference between sampling types in this graphical form was taken from [9]. As you can see, stratified samples are at consistent intervals while perturbation causes the samples to 'shift' inside their bins.



VI. Hierarchical Sampling

Hierarchical sampling is another special step used by NeRFs. NeRFs can be implemented in two main ways. The first is only a coarse detail network that only uses stratified sampling to generate the image. The second approach consists of two networks: a coarse network followed by a fine detail network. The coarse network's outputs are then used to perform hierarchical sampling. Those samples are then passed into the fine detail network to render the image.

To understand hierarchical sampling it is important to understand that only a small fraction of the samples during ray marching are used to render the image. Most of the samples are either in empty space (having an opacity weight of zero) or are occluded by the surface we are rendering and are blocked from contributing to the color of the pixel. To resolve this, hierarchical sampling more heavily samples areas along the ray that we expect to contribute to the pixel's color.

In order to predict which areas along the ray are more heavily sampled, the coarse detail network also outputs the opacity weights for each ray it processes. The weights are then normalized to create a probability density function along the ray[1]. We then sample along

the ray again using this PDF to inform where along the ray we sample. As a result, more samples are taken from locations with higher probability of contributing to the pixel color. The hierarchical samples are then combined with the stratified samples and are all passed into the fine detail model.

VII. Positional Encoding

It has been shown in research that multilayer perceptrons favor low frequency function mappings over high frequency function mappings.[1] This is typically fine as it is preferred to generalize the model and not overfit the data. In the case of image generation, this causes the MLP to struggle to capture fine details. Positionally encoding the input before it is passed to the model for rendering encourages NeRF's MLP to map a high frequency function [1] [8]. To achieve positional encoding, inputs to the model are first passed through a fourier feature mapping.

Intuitively, this can be explained in terms of kernel regression. Kernel regression is a method for fitting a continuous function to a set of data points. A wider kernel corresponds to a lower frequency function and a thinner kernel corresponds to a higher frequency function. Training a fully connected MLP with gradient descent has been shown to be the same as kernel regression as the width of each layer goes to infinity[8]. In other words, we can use the positional encoding of inputs with a fourier feature map to adjust the kernel size of the MLP regression and in doing so select the best kernel size for out data[8]. Interestingly, presentation [8] demonstrates that this opens up new opportunities for MLPs to be applied in other contexts. For example, [8] shows that an MLP with positional encoding is capable of replacing the convolutional generator model of a GAN and perform just as well.

VIII. Experiment

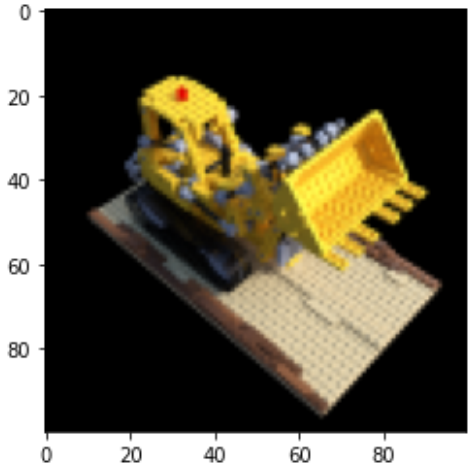
[9] and [10] were used as templates for implementing the NeRF for this paper. [9] provides an in depth tutorial for building a complete NeRF in pytorch. [10]'s model is named 'tiny_nerf' and was written by Ben Mildenhall, the primary author of [1]. Tiny_nerf does not include hierarchical sampling, a fine feature network, or include viewing directions as input to the network. It is designed to be a small implementation that can be trained within minutes instead of more robust models that can take days to train [1]. I expanded 'tiny_nerf' by adding hierarchical sampling, a fine feature network, and editing the layout of the model to test. I also attempted to add directions to the model input but was unsuccessful in making the syntax work.

Over the course of my experiments, I suspected not including camera direction was restricting my model's ability to produce clearer images. I confirmed this was not the case, as I ran [9]'s model which uses camera direction as input and found that its images did not provide any additional clarity. I do not think that the camera direction is relevant for this dataset because all camera directions point at the center of the model. In other words, the training and test images are arranged in a dome around the object. By simply knowing the image's location, the viewing direction can be inferred.

The dataset I used was the Lego bulldozer dataset for [10]. This data set is commonly used in NeRF projects as its images are a manageable resolution of 100 x 100 pixels. The images are of a computer generated model bulldozer built out of Legos. The data is also available as a .npz file which greatly simplifies accessing the data for testing.

The dataset consists of 106 images at camera positions and directions. Similar to other implementations such as [9], I used the first 100 images for training and withheld the last images for comparison. I experimented with adjusting the model's MLP architecture such as its width, depth, activation functions used, and with dropout added. I also compared training the NeRF with and without positional encoding to quantify results.

Depicted below is the original version of image 101 in the dataset.



The standard layout for the fine and coarse models used for testing is depicted below.

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 39)]              0

 dense (Dense)               (None, 256)               10240

 dense_1 (Dense)             (None, 256)               65792

 dense_2 (Dense)             (None, 256)               65792

 dense_3 (Dense)             (None, 256)               65792

 dense_4 (Dense)             (None, 256)               65792

 tf.concat (TFOpLambda)      (None, 295)               0


 dense_5 (Dense)             (None, 256)               75776

 dense_6 (Dense)             (None, 256)               65792

 dense_7 (Dense)             (None, 256)               65792

 dense_8 (Dense)             (None, 4)                 1028

=================================================================
Total params: 481,796
Trainable params: 481,796
Non-trainable params: 0
```

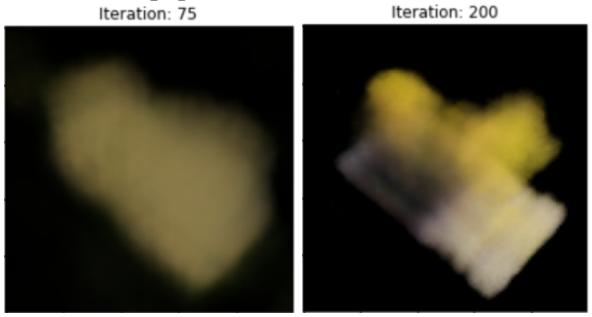
My experiments included measuring PSNR. All implementations I looked at included PSNR as the best way to measure a NeRF's effectiveness. PSNR is Peak Signal to Noise ratio and measures image quality. It represents the ratio between the max possible image value(the original image) and the noise applied to the generated image. This provides a quantitative measure for image quality. More formally it is defined as:

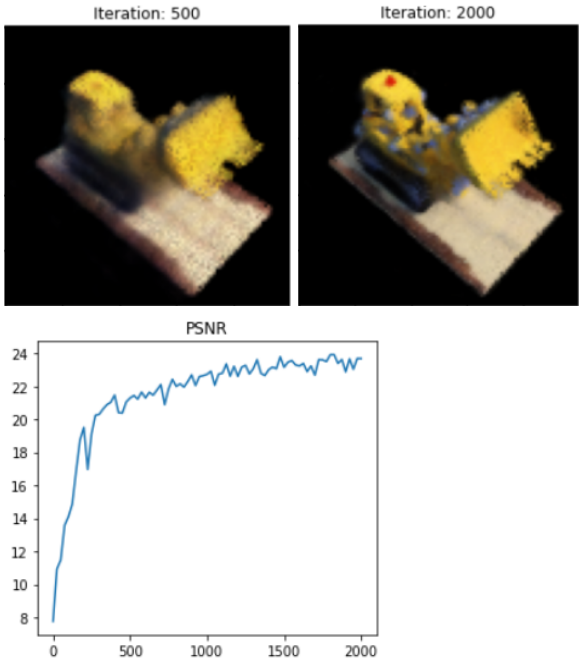
$$PSNR = 20log_{10}(\frac{MAX_f}{\sqrt{MSE}})$$

Where MAX_f is the max signal value for the image (original) and MSE is the mean squared error between the original and our generated image. As the MSE approaches zero the divisor in the PSRN equation decreases and its value increases towards infinity.

IX. Test Results

Depicted below are several points during training the standard NeRF with a coarse and fine model as described in paper [1]:

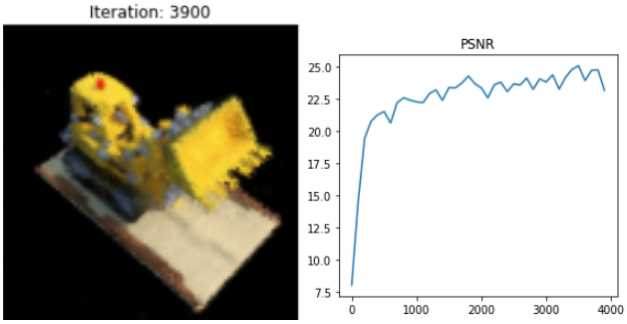




Shown above is a graph of the PSNR during training for image 101 (the implementation of the PSNR graph is from [10]). The graph appears to reach an asymptote at a PSRN value around 24.

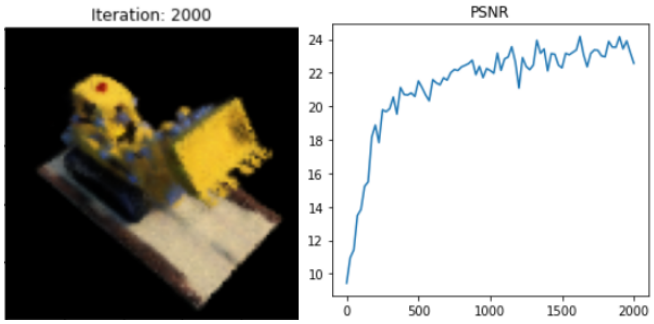
Further experiments focused on improving the clarity of the images generated and improving the PSNR score. My first edit consisted of adding dropout layers after each dense layer in both the coarse and fine models. My hypothesis was that adding dropout layers would force the model to learn new paths for representing the scene and would lead to a higher PSNR value. My initial implementation used 40 percent dropout and this caused the learning to stall. I believe this is because the high dropout rate was actually causing the model to never learn or 'commit' to encoding the scene in one way.

I reduced the dropout to 20 percent and the model no longer stalled but did not improve on image clarity. This test was ran in Google Colab with the session ending after 3,900 training iterations. As shown in the graph below, the training began to asymptote out at around 23 PSNR with a final recorded PSNR value of 23.16.

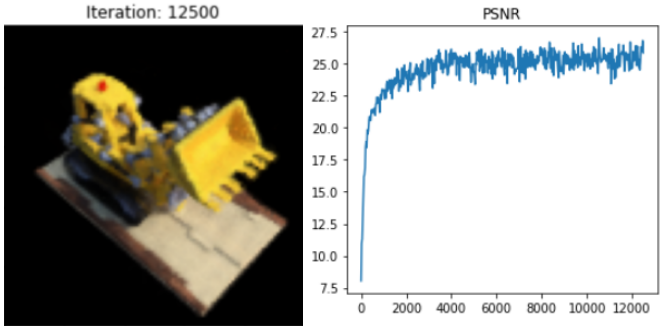


I then tested a model that used Leaky-ReLU activation instead of ReLU. I hypothesized I might be facing the Vanishing Gradient Problem and using a Leaky-ReLU could alleviate this problem. I replaced all ReLU activations with Leaky-ReLUs with 0.3 alpha values. Again, this did not improve results. Other attempts at editing the model to improve performance are listed below:

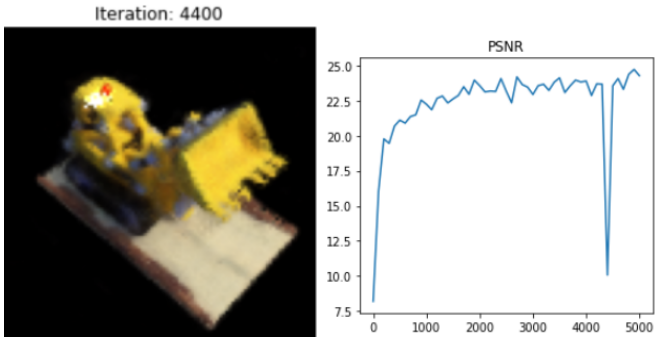
Doubling network depth (16 dense layers instead of 8):



Doubling layer sizes from 256 to 516:



Below is the result of adding a dropout of 20 percent with 0.2 alpha Leaky-ReLUs. The image included below is the image at iteration 4400 when the PSNR value dives down to 10. I believe this was caused by the layer dropouts happening to disable nodes particularly important to rendering the roof of the bulldozer. Looking at the image, there is a section of the roof that is white as if a section of the image was simply not rendered.



To test whether better results could be attained by simply training longer, I tested running the standard NeRF model as described in the paper for 100,000 iterations. The PSNR value never surpassed 25.4.

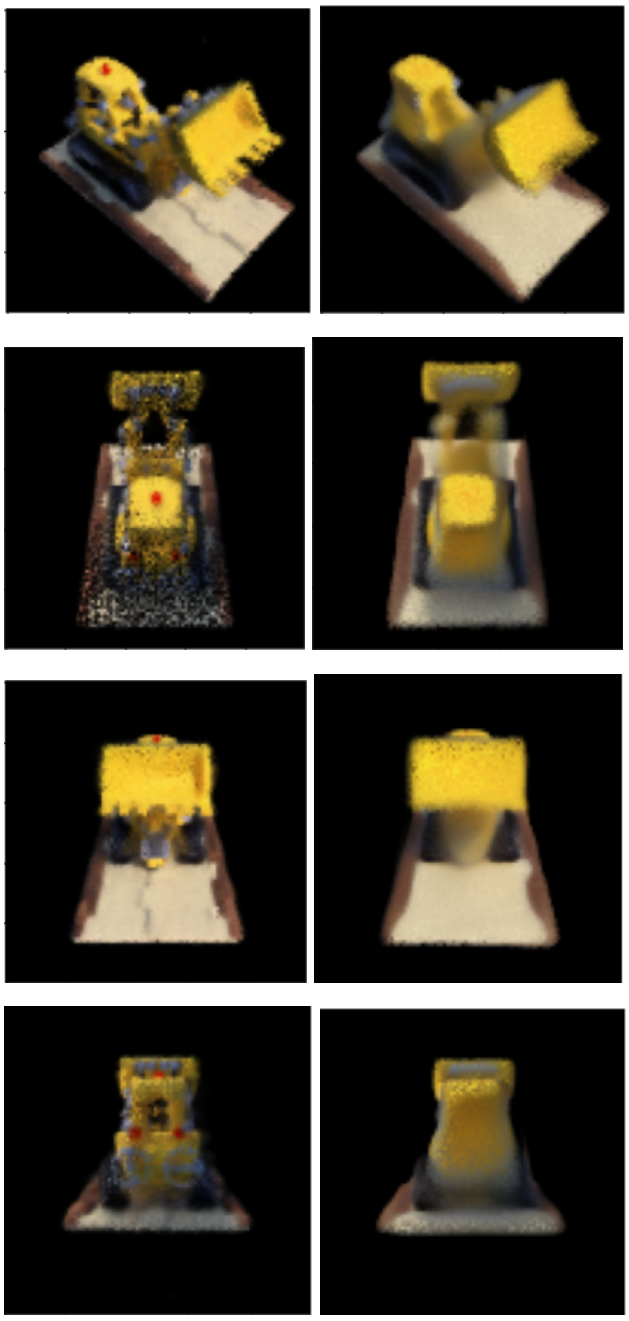
I am suspicious that my results would have continued to improve if given longer training despite the leveling off of PSNR value. Looking at other papers, many trained for several days. I tested for several hours using Google Colab's GPU processor until exhausting service access to reach 100,000 iterations. Paper [1] cites that their training ranged from 100 to 300k iterations to converge on a scene using an NVIDIA V100 GPU. I expect that with additional training, this model would achieve the same clarity as those cited in the associated literature.

Additionally, I think PSNR gives a good quantitative indication of image quality but does not always correspond well to our qualitative understanding of how much better one image is to another. The comparisons between using positional encoding below reflect that discrepancy.

To quantify the benefit of positional encoding on the ability of the network to learn fine details, I ran the standard NeRF for 3,000 iterations once with positional encoding and once without. The PSNR values for four images not included in the training set are listed below.

Image	Positional Encoding	No Positional Encoding
101	23.22	21.47
102	19.79	23.03
103	25.43	22.67
104	27.98	24.71

Images 101, 102, 103, and 104 are depicted below. The left images are the output of the network trained with positional encoding and the right images are from the network without positional encoding.



Comparing the images qualitatively, it is clear that positional encoding allows the network to capture fine details. This is very obvious in the second network not capturing the teeth of the bulldozer or its red lights.

Interestingly, the non-positional encoder network rendered an image with a higher PSNR value for image 102. Looking at the images, it is clear that the positional encoder network is still learning to render the volume at that position. Even though sections have black pixels over them, the image still has more detail and could be said to be a qualitatively better image. This suggests that although PSNR is a good quantitative measure, it does not necessarily always align with our qualitative understanding of which image is 'better'.

X. Conclusions

NeRFs, greatest strengths appear to be its focus on learning the light radiance field, hierarchical sampling, and using an MLP with positional encoding to model the radiance field. Our testing showed how impactful simply preprocessing the input with positional encoding improved results. I found it interesting that despite various changes to the MLP architecture, no noticeable change in performance could be attained. It is possible that using such small resolution images for training and rendering makes the differences between the generated and true images more noticeable. In future work, I intend to train the NeRF with high resolution datasets to see their performance. I expect that the generated model will still have loss from the source images but that loss will be less perceptible to a human observer. I also found that although most of the large improvements seem to happen at the beginning of training, having a long training time to very gradually improve can significantly improve the quality of generated images to the observer.

Researching NeRFs also put their application into a realistic context despite their popularity. For example, the standard NeRF cannot handle scenes that are not static as outlined in paper [12]. I also think that although NeRFs can learn fine details about a scene, they might not be quick enough for real time autonomous systems. Perhaps one of the most exciting insights I gained from this project is that MLPs can be greatly enhanced using positional encoding. I would like to see how MLPs with this enhancement could be applied to other models such as replacing the generative model in a GAN.

References

[1] Mildenhall, B., Srinivasan, P. P., Tancik, M., Barron, J. T., Ramamoorthi, R., & Ng, R. (2020). Nerf: Representing scenes as neural radiance fields for view synthesis. *Computer Vision – ECCV 2020*, 405–421. https://doi.org/10.1007/978-3-030-58452-8_24.

[2] D. Maturana and S. Scherer, "VoxNet: A 3D Convolutional Neural Network for real-time object recognition," 2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2015, pp. 922-928, doi: 10.1109/IROS.2015.7353481.

[3] Lombardi, S., Simon, T., Saragih, J., Schwartz, G., Lehrmann, A., & Sheikh, Y. (2019). Neural volumes. *ACM Transactions on Graphics, 38*(4), 1–14. https://doi.org/10.1145/3306346.3323020

[4] Eslami, S. M., Jimenez Rezende, D., Besse, F., Viola, F., Morcos, A. S., Garnelo, M., Ruderman, A., Rusu, A. A., Danihelka, I., Gregor, K., Reichert, D. P., Buesing, L., Weber, T., Vinyals, O., Rosenbaum, D., Rabinowitz, N., King, H., Hillier, C., Botvinick, M., … Hassabis, D. (2018). Neural scene representation and rendering. *Science, 360*(6394), 1204–1210. https://doi.org/10.1126/science.aar6170

[5] Sitzmann, V., Thies, J., Heide, F., NieBner, M., Wetzstein, G., & Zollhofer, M. (2019). DeepVoxels: Learning persistent 3D feature embeddings. *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. https://doi.org/10.1109/cvpr.2019.00254

[6] Vincent Sitzmann, Michael Zollhöfer, and Gordon Wetzstein. 2019. Scene representation networks: continuous 3D-structure-aware neural scene representations. Proceedings of the 33rd International Conference on Neural Information Processing Systems. Curran Associates Inc., Red Hook, NY, USA, Article 101, 1121–1132.

[7] Tancik, Matthew, Pratul Srinivasan, Ben Mildenhall, Sara Fridovich-Keil, Nithin Raghavan, Utkarsh Singhal, Ravi Ramamoorthi, Jonathan Barron, and Ren Ng. "Fourier features let networks learn high frequency functions in low dimensional domains." *Advances in Neural Information Processing Systems 33* (2020): 7537-7547.

[8] *Fourier Features Let Networks Learn High Frequency Functions in Low Dimensional Domains (2020) YouTube.* YouTube. Available at: https://www.youtube.com/watch?v=h0SXP6lJxak&t=774s (Accessed: November 30, 2022).

[9] McGough, M. (2022, April 28). *It's nerf from nothing: Build a vanilla nerf with pytorch.* Medium. Retrieved December 3, 2022, from https://towardsdatascience.com/its-nerf-from-nothing-build-a-vanilla-nerf-with-pytorch-7846e4c45666

[10] Mildenhall, B. (2021, November 30). *Nerf/tiny_nerf.ipynb at master · bmild/nerf.* GitHub. Retrieved December 3, 2022, from https://github.com/bmild/nerf/blob/master/tiny_nerf.ipynb

[11] Barron, J. T., Mildenhall, B., Verbin, D., Srinivasan, P. P., & Hedman, P. (2022). MIP-Nerf 360: Unbounded anti-aliased neural radiance fields. *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. https://doi.org/10.1109/cvpr52688.2022.00539

[12] Martin-Brualla, R., Radwan, N., Sajjadi, M. S., Barron, J. T., Dosovitskiy, A., & Duckworth, D. (2021). Nerf in the wild: Neural radiance fields for unconstrained photo collections. *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. https://doi.org/10.1109/cvpr46437.2021.00713