LaTeX NeRF Stress Test

Daniel Morton
dcmorton@stanford.edu

Abstract

This project analyses the strengths and weaknesses of the NeRF model for 3D reconstruction. It shows that NeRF works best for small objects and scenes, and that its performance degrades as the size and complexity of the scene being modeled increases.

All the tests of NeRF were done using the original NeRF codebase on GitHub. Some ancilarly code has been added to https://github.com/DanielMorton/Pondhawk.

1. Introduction

Neural Radience Fields (NeRF) [6] the the latest in a series of methods designed to produce novel 3D renderings of objects or scenes. Most previous methods have either used voxels or triangular meshes. More recent techniques have used CNNs to predict point density or color. NeRF replaces the CNN with a multilevel perceptron (MLP), a fully connected neural net that predicts the color viewed by each pixel in a novel camera view.

Despite being introduced with much hype, NeRF has some limitations, especially when it comes to reconstructing scenes. Most of the forward facing scene images used in testing were inheritted from LLFF, and adhere to a rigid pattern of image collection. This article will explore what happens when that pattern is broken. We shall see that, like most models, NeRF is very dependent on the quality of the input data. While the basic strategy of NeRF shows promise, and is successful in small scale reconstruction, in its current form it is a very fragile system.

2. Related Work

Traditionally, 3D reconstruction has been done with Voxels [3] or triangular meshes. More recently, it has become yet another fertile area for deep learning. Early work attempted to model 3D images as level sets of a function approximated by an MLP [1] [7]. Later, 3D occupancy fields [2] , a network that predicts the probability that any point is occupied by an object, were developed to reconstruct the shape of an object. A second network, predicting the texture of the object, could be used to store data about color.

There are several alternative techniques for 3D rendering. Neural Volumes [4] can construct novel views of an object that lies entirely within a bounded volume, similar to NeRF's 360 reconstructions. Neural Volumes uses a 3D convolutional network to predict color and density. Scene representation networks [9] use an MLP to reprensent each coordinate as a feature vector and a recurrent network to aggretate the colors along the ray. Local Light Field Fusion [5] uses a 3D CNN to predict RGBα for each input view and produces novel views by composing α's and blending the images.

3. Background

Neural Radience Fields is the latest in a series of techniques that represent an object or scene as the weights of an MLP, a deep neural network composed entirely of fully connected layers. For these purposes, an object can be viewed as having two components, its physical geometry (the location of all its componet points in space) and the color of each point (a function of both the intrinsic materials of the object and the way light reflects off of the object). The model then needs to have two sets of inputs, a 3D coordinate (x, y, z) specifying a point in the scene, and a two dimensional direction (θ, ϕ) (in practiced specified by the three dimesnional vector $\mathbf{d}$) specifying the direction from which the point is being viewed. The output is a three dimensional color vector (r, g, b) and a one dimensional density vector σ.

This is, of course, not what the eye or cameras sees. The color registered by each camera pixel is accumulated color picked up along the ray leading to that pixel. Theoretically, this is expressed as integral over all the color along the ray, where $\mathbf{r}(t)$ is the ray and $\mathbf{c}$ and σ are the color and density functions respectively. The total color of ray $\mathbf{r}(t) = \mathbf{o} + t\mathbf{d}$ is expressed in the integral below.

$$C(\mathbf{r}) = \int_{t_n}^{t_f} e^{-\int_{t_n}^{t} \sigma(\mathbf{r}(s))ds} \sigma(\mathbf{r}(t)) c(\mathbf{r}(t), \mathbf{d}) dt$$

A scene is composed of all the rays that enter a camera positioned in specific location. If a scene has $height \times width$ pixels, then there will be $height \times width$ values of C computed. The MLP, however, only predicts $\mathbf{c}$; NeRF needs to estimate C by sampling $\mathbf{c}$ as points along the ray.

The naive method of estimating an integral is to replace it with a weighted sum of evenly sampled points along the interval of integration. That would be less than desirable in this case since most points along the ray will lie in empty space or behind the surface of the object and thus contribute nothing. In practice, along one ray, almost all of the color almost all of the time will come from one point on the object surface. Learning where this point is part of the MLP training. But in order two train the MLP, we need to sample the points along the rays.

In order to get out of this chicken-and-egg situation, NeRF uses two models. The first "coarse" model is trained using an even sample of 64 points along each ray. The density estimates of this model are then used to estimated a CDF that is then used to resample points along the ray (64 or 128 depending of model parameters) with most points sampled at the regions of highest density. This new set of points is then used to train a more accurate "fine" model. Only the outputs of the fine model are used for prediction but training is done on both models in order to compute the densities along the rays.

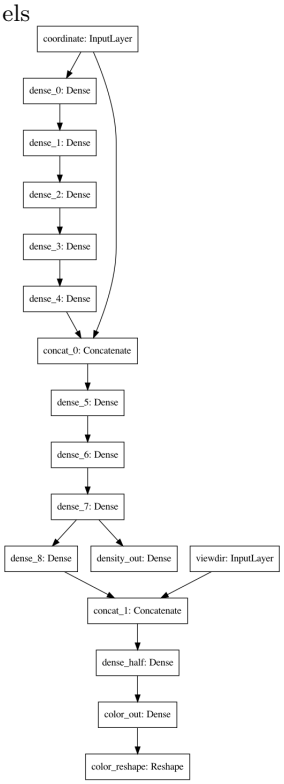
The training loss is surprisingly simple; merely the sum mean squared errors of the coarse and the fine models over all the pixels in the training images. If $\hat{C}_c$ and $\hat{C}_f$ are the predictions of the coarse and the fine model, the loss in symbols is

$$\sum_{r \in \mathcal{R}} \left\| C(r) - \hat{C}_c(r) \right\|^2 + \left\| C(r) - \hat{C}_f(r) \right\|^2$$

where $\mathcal{R}$ is the set of all rays in the training set. But since the MLPs themselves only produce estimates at points along the rays, the formula connecting C and $\mathbf{c}$ makes backpropagation that much more complicated.

There is preprocessing that must be done on the input as well. Although MLPs are know to, theoretically, be universal approximators, in practice there are some functions they can approximate better than others. One set they do particularly poorly at is sinusoidal functions. Since the earliest days of digital image processing, it has been known that images naturally decompose into a Fourier basis. To aid reconstruction,

Figure 1. The architecture of the deep fully connected models



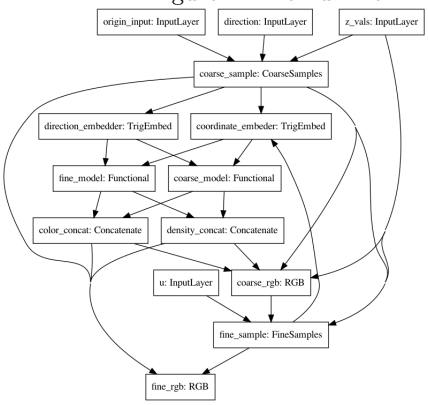
we want to include these features in the model. Since MLPs cannot approximate high frequency sinusoids, but sinusoid functions exist in every standard math package, we can simply add the sinusoids as inputs. For each coordinate of each point, the model accepts as inputs

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

The value $L = 9$ is usually chosen to match the Nyquest freqency for the image resolution. For higher resolution images, a larger value could be used. The same type of embedding is used for the coordinates of the direction vector $\mathbf{d}$, although a value of $L = 3$ is usually considered adequate. The end result is that the MLP accepts 63 coordinate inpute values and 27 direction input values.

The full MLP consists of eight fully connect layers taking data from a coordinate input layer. There is a skip connection from the input layer to the sixth layer. The output of the eighth layer goes to a softmax layer that predicts the density contribution of that point. The eight layer also outputs to a ninth dense layer that, concatenated with the direction input, feeds into one more dense layer followed by a dense sigmoid layer that predicts the color contribution in the given direction.

Figure 2. The Full NerF Model



All the dense hidden layers have 256 nodes and ReLU activation except for the final dense layer before the color output (dense_half), which has 128 nodes, and dense_8, which has a linear activation. Density is only determined by position; color is determined by position and direction.

Training is done on batches of individual rays. Each ray is sampled at 64 points for input for the coarse model and an additional 64 or 128 points for the fine model. A single training example consists of 128 or 192 input points and three output point, the (r, g, b) values. In effect, althoug never constructed explicitly, this amounts to two siamese networks, one with 64 copies of the same model the other with 64 or 128 copies of the same dense MLP.

This can be modeled as one graph in TensorFlow. See image 4. The origin and direction inputs are the camera focal point and ray directions for all the rays associated with all the images. This data is extracted from the camera poses. The coarse and fine sampling layers produce point sampling along each ray. The trigonometric embedding were described above. The layers labeled z_val and u are ancilarly inputs needed to make the random ray sampling word. The concatenate layers combine the outputs of the points sampled along the rays. And the RGB layers do the integral approximation.

Apart from the two functional layers, none of the layers in this model have any training parameters. It is possible to show that this is exactly the same network architecture, in explicit form, as that used in the NeRF codebase in implicit form. The NeRF codebase comes with two sample objects, a lego earth mover and a large potted plant labeled fern (in truth, a palm tree, but we'll ignore that for now.) Along with the training images, these samples come with training parameters and model weights. Since most of my work up to that point had involved forward facing scenes, I decided to look at the "fern". Using the parameters, 64 extra sampling points for the fine layers, a small amount of noise in the random point sampling to make the model more robust, and some normalization of the camera poses for regularity, as well as the model weights inserted to the model shown in 4, I confirmed that 4 produced the same outputs for both the coarse and fine predictions at the implicit model in the NeRF GitHub repositiory. From this I could conclude that both models performed identical forward propogation.

Unfortunately, the explicit model in Tensorflow Keras proved impossible to train. Possibly because it is a 192-fold siamese network, backpropogation turned all weights and biases to *nan*. Reconfiguring the network, dropping the, already low, learning rate, and even gradient clipping were to no avail.

All subsequent experients were done using the original NeRF code.

4. Approach

"He that breaks a thing to find out what it is has left the path of wisdom." Perhaps Gandalf was right. Nonetheless, in order to understand NeRF well enough to work with it, I had to break it. The first task was to understand the input data. NeRF requires two sets of data, a collection of images of an object or scene and the relatvie camera poses between these images. Collecting images is easy, one just needs a camera and all phones have those these days. I took some images with my Pixel but most were taken with a Nikon 3300 DSLR camera.

Compiling the relative camera poses is a bit harder. Basic Structure from Motion algorithms are simple enough mathematically, but require a certain amount of tuning, especially at scale. NeRF does nothing to compute poses, relying on the COLMAP [8] software also used by LLFF. COLMAP produces sparse representations of 3D scenes using standard SfM algorithms in a three stage process. First, SIFT features are extracted. The second, and frequently longest, stage is feature matching. Exhaustive matching (all pairs of images are compared) is $O(N^2)$ but is the most thorough option and not prohibitive for the types of image collections used by NeRF. (Budget and hour for 100 images, and this step only needs to be done once.) The final sparse reconstruction step computes the camera poses. This last step is very sensitive to initial conditions, and in extreme cases will not produce a sparse reconstruction. After much experimenting I fould that setting the parameters multiple_models to true (i.e. don't assume the same intrinsic camera matrix for all images) and minimum initial triangulation angle at 4 worked in most situations.

Once both sets of inputs are available, then it is time to train the model. Since the purpose of this exercise is to test how well NeRF can reconstruct more complicated scenes than those shown in the paper, I saw no reason to tamper with the default parameters. The initial learning rate is kept at 5×10^{-4} decaying to 5×10^{-5} over $250K$ steps. Since 64 extra samples are used for the fine model in the fern/palm example, I continued to use 64 samples in my experiments. Early experiments had been done with raw_noise_std (a parameter to add randomness to the density values) set to zero, but this produced strange floaters (components of the image that detached and seemed to float in space) so I reverted to the default noise of 1. Since both the phone and the camera produce fairly high resolution images, I also kept the default downsample rate of 8. Every eighth image is reserved for validation.

NeRF reconstructs scenes under two separate circumstances. The first is forward facing scenes; the same type LLFF reconstructs. The second is reconstruction of objects from 360 camera views where the object can be convienently placed in a 3D coordinate grid. In both cases, there are some adjustments that have to be made to the camera poses so that the model can treat the scene or object as being within certain bounds. The two situations have different adjustments; in both cases it is wise to the defaults. The one time I tried training NeRF on a 360 scene with forward facing defaults the result was surrealist art.

Since the goal of the project is to stress test NeRF it seemed natural to try training models of large complicated scenes. Since the 360 reconstructions had only been tried on small objects, I focused on forward scene examples. The results never looked as good as those presented in NeRF, so I began to scale back to smaller scenes. That had mixed success, which led to reexamination of the poses used in the original NeRF paper. In every case, the camera poses were in a tight grid pattern and all facing forward. Crucially, the main part of the scene was in every picture.

5. Experiments

This led to a natural two part experiment to evaluate NeRF's abillity to represent large complex scenes. My outside porch 3 has exactly the sort of repeating rail pattern that makes for good computer vision test fodder. I took a collection of 180 pictures of the porch from every angle and distance I could get an unobstructed view. If sheer volume of images were enough to get a good 3D reconstructions, I could not possibly have done better.

The second part was to take pictures of just one subset of the porch, the railling to the right of the main stairs. This I further chopped into two blocks, the parts left and right of the flower box. I took a 5 by 4 5 grid of images of the left side, and a 5 by 8 6 grid of image of both sides (with another three pictures added for good measure.) NeRF models were constructed for the 180 picture set, the 5 by 4 grid, and a combination of the 5

Figure 3. Side Porch

Figure 4. Camera Positions from Porch Photograpy

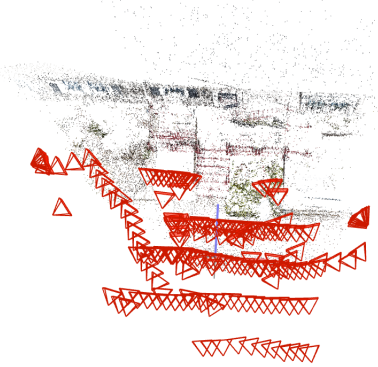


Figure 5. Twenty Image Camera poses

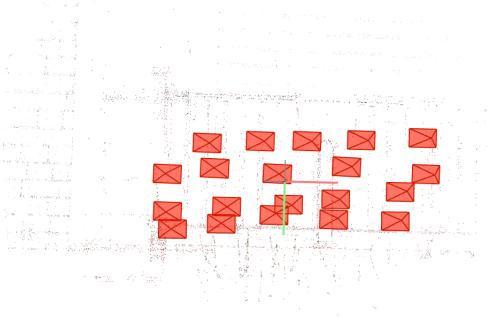


Figure 6. Sixty-Three Image Camera poses

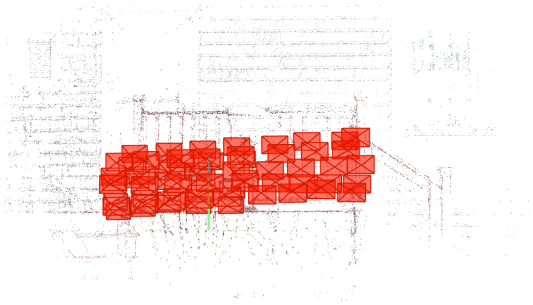


Table 1. Porch Test Mean and SD

	PSNR	MS-SSIM
20 Image Porch	18.40 ± 2.14	0.8512 ± 0.0628
63 Image Porch	15.63 ± 2.58	0.6396 ± 0.1511
180 Image Porch	15.47 ± 1.54	0.6112 ± 0.1423

by 4 and the 5 by 8 grids (making the third image set a strict superset of the second.)

The overall results were the exact opposite of what one might have expected. The 20 image collection had the best reconstruction results, with a PSNR of 18.39 and an MS-SSIM of 0.8512 on the validation set. The 63 image superset had PSNR of 15.63 and MS-SSIM of 0.6396. The bigger surprise came with the complex 180 image set; it performed only slightly worse on it's validation set than the 63 image set, with PSNR of 15.49 and MS-SSIM of 0.6112. Although NeRF's ability to reconstruct a scene clearly degrades as the image collection becomes more complex, the dividing line is not where the arangement of camera poses becomes more complex. In the 20 picture case, the intersection of the images is non-empty. In both the 63 and 180 image examples, there is no point that appears in all the images. It seems likely that this dividing line is where NeRF stops having a region common to all the input images.

The difference in reconstruction quality becomes painfully obvious when comparing the same scene reconstructed by both models. The model constructed on 20 images produces an almost perfect reconstruction 7 of the porch, only struggling with the twigs of the bush underneath. A casual observer could easily mistake it for a real photograph. The model constructed from the larger image set produces something not dissimilar to a low dimensional PCA approximation 8.

A comparable reconstruction from the 180 image reconstruction looks like a photo taken from a moving vehicle 9, failing to reconstruct the bush in front of the

Figure 7. 20 Image Porch Reconstruction

Figure 8. 63 Image Camera Reconstruction

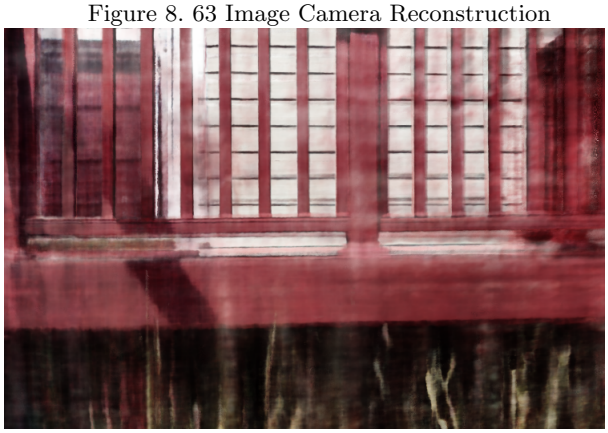


Figure 9. 180 Image Camera Reconstruction

porch at all. And that's one of the better scenes.

A second, albieit simpler, experient produced the same results. A full bookshelf provides a surprisingly complex scene in a small environment. A simple way to test the hypothesis that NeRF only does well when the training set has a non-empty intersection would be to use NeRF to model one shelf, and then produce a second model of two shelves (one above the other) where the images where each image is of one shelf. A row of images between the two shelfs is added to ensure

Figure 10. One Shelf Camera poses

Figure 11. One Shelf Camera poses

Table 2. Shelf Test Mean and SD

	PSNR	MS-SSIM
One Shelf	23.50 ± 1.52	0.910 ± 0.0244
Two Shelf	22.31 ± 1.65	0.8281 ± 0.0479

NeRF doesn't try to model two disconneted regions.

The one shelf reconstruction 10 used 68 images and the two shelf reconstruction **??** used 64. All were taken at close range to avoid any other household objects confusing the model. The results were as expected. The one shelf reconstruction had test set PSNR of 23.50 and MS-SSIM of 0.9102 while the two shelf reconstruction had PSNR of 22.31 and MS-SSIM of 0.8281. It's worth noting that the worse reconstruction in this case has MS-SSIM score closer to the best porch reconstruction. Since this was an inside experiment and the images were taken at closer range, we can expect reconstructions to have better scores regardless of other factors.

Once again, the difference in reconstruction quality becomes apparent when looking at similar scenes. The one shelf model can even reconstruct iridescent surfaces 12 and fairly small typefaces 13with good success. The two shelf model struggles with both 14 15.

The two shelf set of book images provided an opportunity to test how fine the "fine" model in the NeRF reconstruction needed to be. Aside from the usual 128 point fine model, I also tested a 192 point fine model (128 extra points sampled along each ray.) The results were surprising, both the PSNR and MS-SSIM

Figure 12. Iridescence Reconstruction on One Shelf

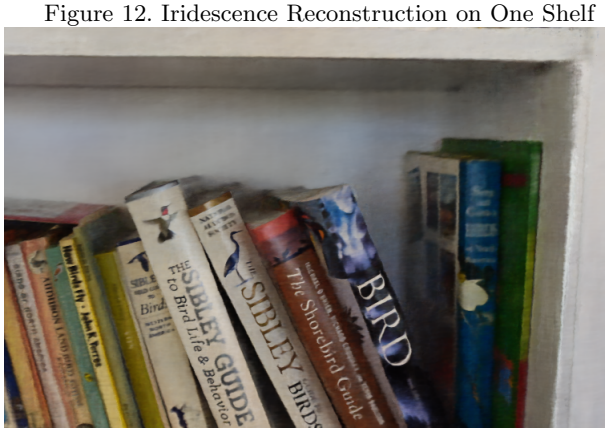


Figure 13. Fine Typeface Reconstruction on One Shelf

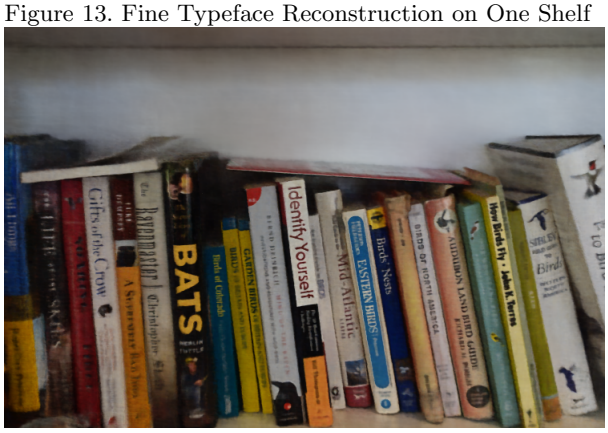
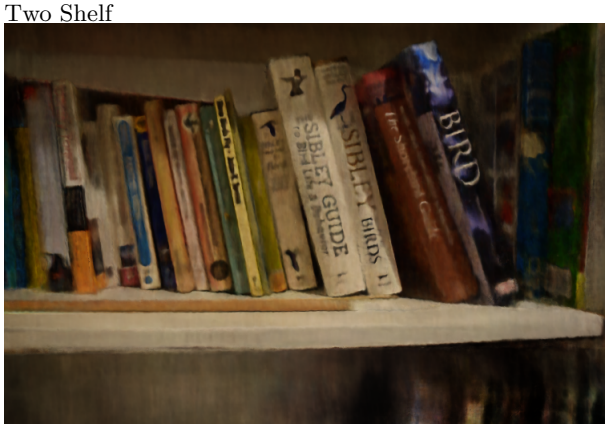


Figure 14. Iridescence)or lack thereof) Reconstruction on Two Shelf



measurements were slightly lower for the finer model, dropping to 22.25 and 0.8207 respectively.

It is worth making a note about training and validation at this point. The NeRF paper claims that between $100K - 300K$ iterations are normally needed to train a NeRF model. While it is true that training error will continue to improve for that period, it is often the case that the test error will plateau at around $50K$ or even $20K$ iterations 16 17. Conveniently, these

Figure 15. Typeface Reconstruction on two Shelf

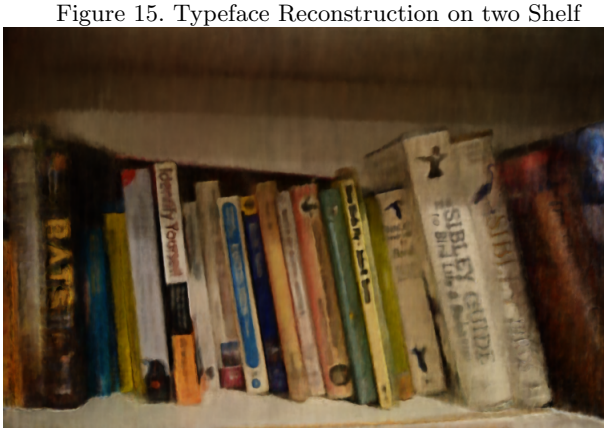


Figure 16. Two Shelf Validation Convergence

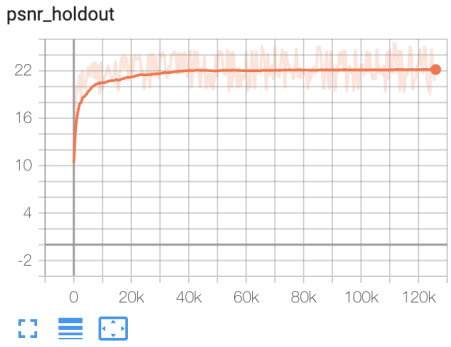
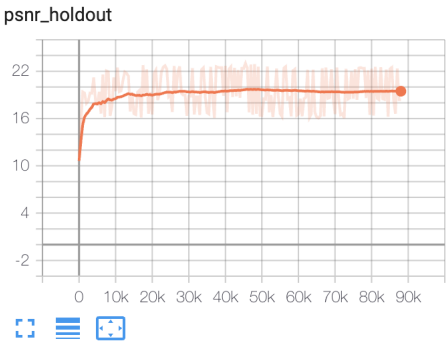


Figure 17. 63 Image Porch Validation Convergence

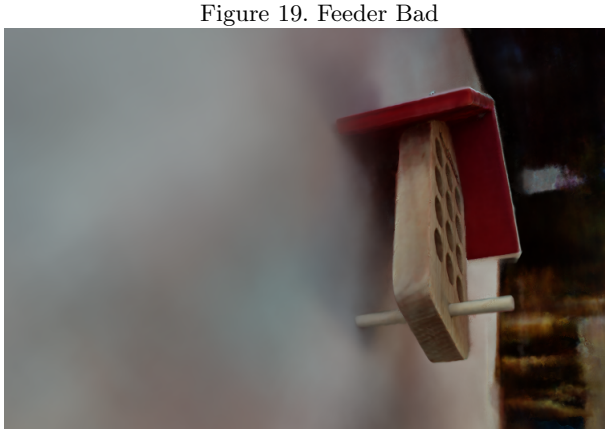


are ususally the examples where the reconstruction is performing poorly; if it looks like the test PSNR has leveled off early, there is a good chance the model will not perform well.

Although most of my work was with forward facing scenes, I did have one successful 360 reconstruction. There was no experiment here in the sense that I was not comparing reconstruction settings, but there are a few interesting observations. In order to make an accurate reconstruction, I needed to take a number of pictures, 80 in all, in a series of arcs around the object; in this case a bark butter bird feeded. This is

Figure 18. Feeder Good

Figure 19. Feeder Bad



a fairly small object, which ment the reconstruction was quite good, but the evaluation metrics had much higher standard deviations than those for the forward facing scenes. PSNR had a mean of 23.88 but with an SD of 8.18, while MS-SSIM had a decent 0.8281 mean, the SD was nearly a quarter of that at 0.1958. Some, but not all, of that variation may come from outside elements that intrude on the scene from some angles 19.

6. Conclusions

While NeRF performs well in a limited range of range of reconstructions, its value as a general purpose tool remains limited. It is not difficult to break. The good news is that the number of images is not particularly relevant; NeRF can reconstruct a scene from a little as twenty. The method of image collection is important. If an object or scene is to be the subject of a NeRF reconstuction, images need to be collected in a very methodical way.

From a computational standpoint, it appears that 64 extra sampling points for the fine model are adequate, and anything more may cause overfitting. It would be

worthwhile to do side-by-side comparisions of the two models with other objects.

A possible follow-up to the porch experiments would be to take a fourth set of images the full right side of the porch, but from a slightly greater distance. That way, the full right side of the porch could be modeled by images that have a nonempty intersection. Most, likely this would produce a reconstruction with an accuracy more in line with the 20 image porch model. It would also be interesting to see if adding the 63 closer range porch images has any effect on the reconstruction.

At one point I had ambitions of trying to reconstruct a car using NeRF and 360 collected images. Now I think that either the number of images would be prohibitive or that the images would have to be taken from an impractical distance. For the moment, it seems 360 reconstructions with NeRF are best left to small objects.

A common pattern among the poorly reconstructed images is their resemblance to low dimensional PCA or DCT reconstructions. This may be a factor of the number of nodes in each fully connected layer. Doubling the number of nodes, 512 instead of 256, may result is a higher resolution approximation of large scenes.

References

[1] Jiang, C., Sud, A., Makadia, A., Huang, J., Nießner, M., Funkhouser, T.: Local implicit grid representations for 3d scenes. In: CVPR (2020)

[2] Genova, K., Cole, F., Sud, A., Sarna, A., Funkhouser, T.: Local deep implicit functions for 3d shape. In: CVPR (2020)

[3] Kutulakos, K.N., Seitz, S.M.: A theory of shape by space carving. International Journal of Computer Vision

[4] Lombardi, S., Simon, T., Saragih, J., Schwartz, G., Lehrmann, A., Sheikh, Y.: Neural volumes: Learning dynamic renderable volumes from images. ACM Trans- actions on Graphics (SIGGRAPH) (2019)

[5] Mildenhall, B., Srinivasan, P.P., Ortiz-Cayon, R., Kalantari, N.K., Ramamoorthi, R., Ng, R., Kar, A.: Local light field fusion: Practical view synthesis with prescrip- tive sampling guidelines. ACM Transactions on Graphics (SIGGRAPH) (2019)

[6] Mildenhall B., Pratul P. Srinivasan, Matthew Tancik, Jonathan T. Barron, Ravi Ramamoorthi, and Ren Ng. NeRF: Representing Scenes as Neural Radiance Fields for View Syn- thesis. ECCV, 2020

[7] Park, J.J., Florence, P., Straub, J., Newcombe, R., Lovegrove, S.: DeepSDF: Learn- ing continuous signed distance functions for shape representation. In: CVPR (2019)

[8] Schönberger, Johannes Lutz and Frahm, Jan-Michael. Structure-from-Motion Revisited. Conference on Computer Vision and Pattern Recognition (CVPR) 2016

[9] Sitzmann, V., Zollhoefer, M., Wetzstein, G.: Scene representation networks: Con- tinuous 3D-structure-aware neural scene representations. In: NeurIPS (2019)