Online Window Shopping with NeRF

Zhengxun Wu
wukey92@stanford.edu

Abstract

When we are shopping online, we always want to know more about the products from the product images, especially on those unofficial e-commerce platforms like Facebook Marketplace where images are usually the only available media. What if we can reconstruct a 3D video out of these images captured from the limited different angles? Wouldn't that be useful to give us a better shopping experience by being able to do window shopping first? Reconstructing unseen views from a limited number of images falls into the category of novel view synthesis, which is a long-standing problem at the intersection of computer graphics and computer vision. Fortunately, with the explosion of Neural Radiance Fields (NeRF), neural networks can be used to tackle this challenging task and can achieve impressive results. However, generating these 3D scenes with high-quality is very computationally intensive for NeRF, which limits the usage of this awesome technique to meet tasks requiring a low end-to-end latency like our use case. In this paper, we fine tune NeRF to find a sweet spot between performance and image quality, and make an application generating 3D video out of 2D product images where images are expected to have an object positioned in the center with relatively simple background.

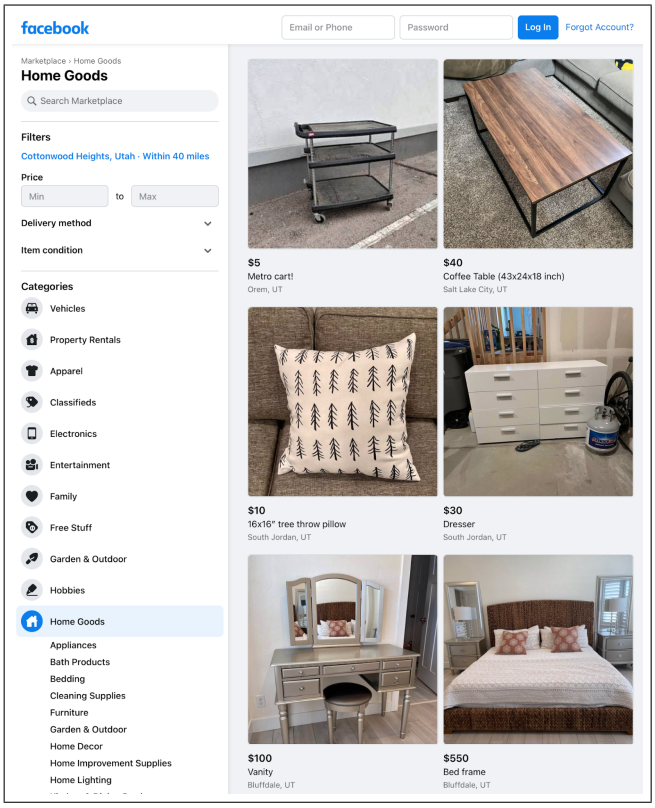

Figure 1. A Screenshot of Facebook Marketplace.

1. Introduction

In this section, we revisit the problem statement and describe our overall plan including model selection and optimization.

1.1. Problem Statement

In this work, we aim to reconstruct 3D video out of a limited number of multi-view 2D images using NeRF[6]. However, the difficulty of the task will vary drastically depending on the attributes of the images (e.g. number of objects, movement, position). To limit the scope of this project, we are restricting our use case to online shopping on non-official e-commerce platforms like Facebook Marketplace where people can post the pictures of the goods they want to sell. Unlike Amazon where seller provide high quality images along with a video or even AR capabilities, on Facebook Marketplace, image is usually the only available media with a very limited number. As for the object in the images, they are usually positioned in the middle with a less complicated background as shown in Figure 1. So we'll try to make an application that can reconstruct 3D video out of 2D images with above attributes to enhance the online shopping experience.

1.2. Model Selection

This problem falls in to the category of novel view synthesis. Volume rendering is an effective technique introduced in [4] regressing a 3D volume of density and color[1]. Then NeRF enhance the idea by taking the DeepSDF archi-

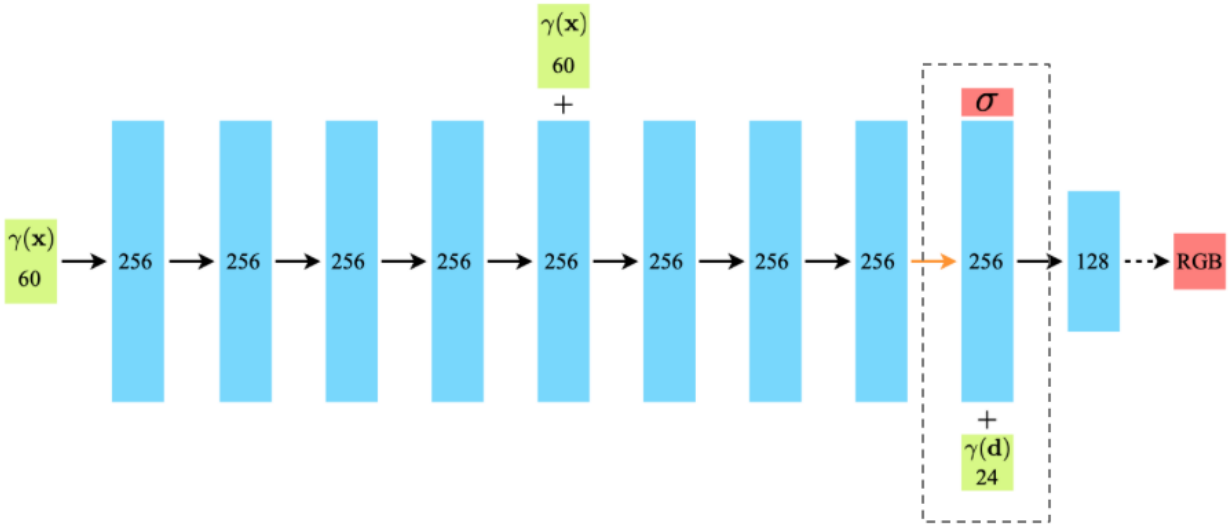

Figure 2. Visualization of the neural network architecture.

tecture but regressing not a signed distance function, but density and color[1], which leads to the simplest but ultimately the most influential model. It's high impact lies in its simplicity but one of the drawbacks comes with it is the poor performance: it's slow for both training and rendering. According to the description in their Github repo, it needs 200K iterations (about 15 hours) to finish training the model and render a high quality video.

1.3. Model Optimization

Obviously, the performance can't meet the requirement of using it for our use case. We plan to do experiments to figure out the best combination of hyper parameters and apply a few optimizations to the neural network architecture to speed it up for particular inputs as described above. NeRF mainly consists of 3 major components plus a multilayer perceptron (MLP). There 3 major components are: 1. view dependence; 2. positional encoding that has a hyper parameter L controlling the degree of higher dimensional space for the input vector; 3. hierarchical volume sampling. There are mainly 2 hyper parameters involved in the MLP and they are: 1. number of training images; 2. number of sampling.

Figure 3 shows the result of an ablation study of NeRF model[6]. We'll use similar strategy to find out the best combination of these components. The evaluation metric we are using here is the same as that used in the NeRF paper: peak signal to noise ratio (PSNR). It's calculated using the following formula:

$$PSNR = -10 * log(Loss) \quad (1)$$

	Input	#Im.	L	(N_c, N_f)	PSNR↑	SSIM↑	LPIPS↓
1) No PE, VD, H	xyz	100	-	(256, -)	26.67	0.906	0.136
2) No Pos. Encoding	$xyz\theta\phi$	100	-	(64, 128)	28.77	0.924	0.108
3) No View Dependence	xyz	100	10	(64, 128)	27.66	0.925	0.117
4) No Hierarchical	$xyz\theta\phi$	100	10	(256, -)	30.06	0.938	0.109
5) Far Fewer Images	$xyz\theta\phi$	25	10	(64, 128)	27.78	0.925	0.107
6) Fewer Images	$xyz\theta\phi$	50	10	(64, 128)	29.79	0.940	0.096
7) Fewer Frequencies	$xyz\theta\phi$	100	5	(64, 128)	30.59	0.944	0.088
8) More Frequencies	$xyz\theta\phi$	100	15	(64, 128)	30.81	0.946	0.096
9) Complete Model	$xyz\theta\phi$	100	10	(64, 128)	**31.01**	**0.947**	**0.081**

Figure 3. An ablation study of NeRF Model.

In the following subsections, we'll discuss the detailed optimization plan for each component.

1.3.1 View Dependence

The input of NeRF is a 5D coordinates that contains location and viewing direction. According to NeRF paper, removing view dependency prevents the model from recreating some lighting effects (e.g. specular reflection). Given that the accuracy requirement is low for our use case, if removing view dependency can significantly improve the performance, it'll be a proper optimization.

1.3.2 Positional Encoding

In addition to adding view dependency into the input vector, NeRF also applies positional encoding to map this 5D input to a higher dimensional space whose degree is controlled by a hyper parameter L. The encoding function is:

$$f(p) = (sin(2^0\pi p), cos(2^0\pi p), ..., sin(2^{L-1}\pi p), cos(2^{L-1}\pi p)) \quad (2)$$

In NeRF paper, L is set to 10 for position vector. We'll try using different values to find out which one can improve the training time while meeting our quality requirement.

1.3.3 Hierarchical Volume Sampling

Hierarchical volume sampling is a rendering strategy used by NeRF during final rendering in order to increases rendering efficiency. The idea is optimizing two networks simultaneously that one for "coarse" sampling and one for "fine" sampling. Thus, during rendering, it's possible to get a more informed sampling of points based on the relevant parts sampled from the coarse network. According to Figure 3, it might be the least import factor towards overall image quality, we'll only explore whether it can speed up training. If not, we'll likely to exclude this component.

1.3.4 MLP

The neural network used by NeRF is as simply as a MLP that mainly consists of 8 fully-connected ReLu layers, each with 256 channels. On the 5th layer, there's a skip connection that concatenates the input to the layer's activation. The positional encoding of the input location is passed through the network.

Figure 2 details the architecture of the fully-connected network. The area marked with dotted lines will be excluded when view dependency is removed.

2. Related Work

In this section, we go through other methods to tackle this novel view synthesis problem before the exploration of NeRF. Basically, there are 2 folds of methods: image-based rendering and feature embedding-based rendering. We analyse these methods to see whether or not they might be a better fit for our use case. Besides, we also take a look at some efforts to optimize NeRF to improve it's training and inference performance.

2.1. Image-based Rendering

2.1.1 Interpolation

The idea of using interpolation technique is quite straightforward, which is mainly about applying a good interpolation technique on a set of input images whose difference is small enough. This kind of methods require a dense sampling of views which makes the rendering super inefficient. Although, LLFF technique is created to reduce the input views by 4000 times [5], it still requires lots of 2D images to generate 360 video.

2.1.2 Feature Embedding

Another way to do image-based rendering is to optimize a convolutional neural network to learn persistent 3D feature embedding. However, CNN is too complicated in terms of time and space which might not be practical for our use case.

2.2. 3D Shape Representation

Another group of methods to tackle novel view synthesis is 3D shape representation. Usually, it needs a deep network being deployed and optimized to map xyz coordinates to latent representation of 3D shapes directly. However, it requires the dataset to have the ground truth of 3D geometry which is hard to obtain.

2.3. NeRF Optimization

Recent works propose a variety of optimizations to mitigate the performance issue of NeRF. The optimization mainly includes three aspects: generality [7], training performance and inference performance [1] [3]. Let's look at the efforts spent on improving training performance particularly, as it will be crucial to the final user experience. JaxNeRF [2] implements NeRF in JAX instead of TF, which speed up training from days to hours. Learned Initialization [8] uses meta-learning to find a good weight initialization for faster training.

3. Technical Approach

As it described in problem statement section, this problem is essentially view synthesis - generating unseen views from seen views. Given that NeRF has the simplest model among all techniques in this area. We are proposing to leverage NeRF to deliver a practical solution to enhance the user experience of the unofficial online shopping (e.g. Facebook marketplace, Craiglist). Due to the limited timeline, we might not be able to integrate other optimization techniques like meta-learning and JAX implementation into the project, but we'll analyse the improvement these techniques can bring to us.

Figure 4 shows the visualization of the overall pipeline and details of each component.

For this project, there are mainly 2 stages: 1. model fine tuning and 2. application development. In the following subsections, we discuss the detailed tasks to be done in each stage.

3.1. Model Fine Tuning

This work is mainly within the training module. We fine tune those hyper parameters listed in Model Optimization section and try out the combinations listed in Table 1.3.4 and find the most efficient one by comparing the PSNR. The reason why we can't leverage the result of NeRF listed

Combination Index	Images	Iterations	Input Dimension (L)	Sampling (N_c)	Has View Dependency
1	100	2000	6	64	true
2	100	2000	10	64	true
3	100	2000	6	128	true
4	25	2000	6	64	true
5	25	2000	10	64	true
6	10	2000	6	64	false
7	5	2000	6	64	true
8	5	2000	6	128	true
9	5	2000	10	64	true
10	5	2000	10	128	true

Table 1. Parameter Combinations.

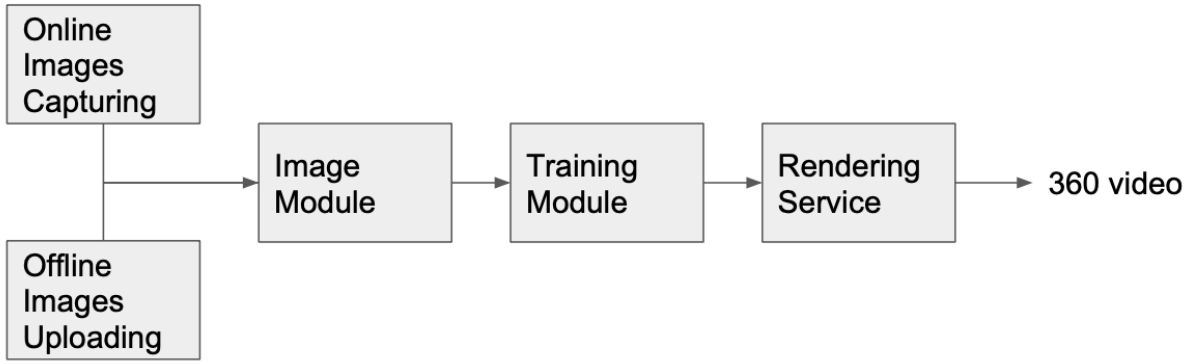

Figure 4. Visualization of Overall Pipeline. There are 2 ways of feeding images to the image module: online image capturing where users take a picture of the object from different angles online and offline images uploading where users upload images taken before. Image module is responsible for processing images to the format required by the training module. Training module is NeRF with fine tuned parameters. After training finishes, the trained model along with input images will be routed to rendering service to generate a 360 video.

Combination Index	PSNR
1	23.5
2	23.8
3	24.2
4	25.0
5	23.8
6	22.5
7	23.0
8	15.0
9	24.0
10	16.0

Table 2. Experiment Result

in paper is because, in order to get a high quality image, the number of images used for training is about 100 which is not practical for our use case where we are likely to access to a limited number images (10). In this project, we use 100 and 25 training images as the baseline and explore the performance downgrade using 10 and 5 images.

By running the pipeline, we find out that the average training speed is 0.25 seconds per iteration. We choose 2000 iterations in the experiment to limit the total runtime to 500 seconds.

3.2. Application Development

Beyond optimizations, we also need to work on the following components to set up an end to end pipeline:

1. Image capturing/uploading module that prepares images for training.

2. Training module that trains the NeRF with found optimal hyper parameters combination.

3. Rendering service that infers unseen views using the trained NeRF and generates a 360 video out of these views.

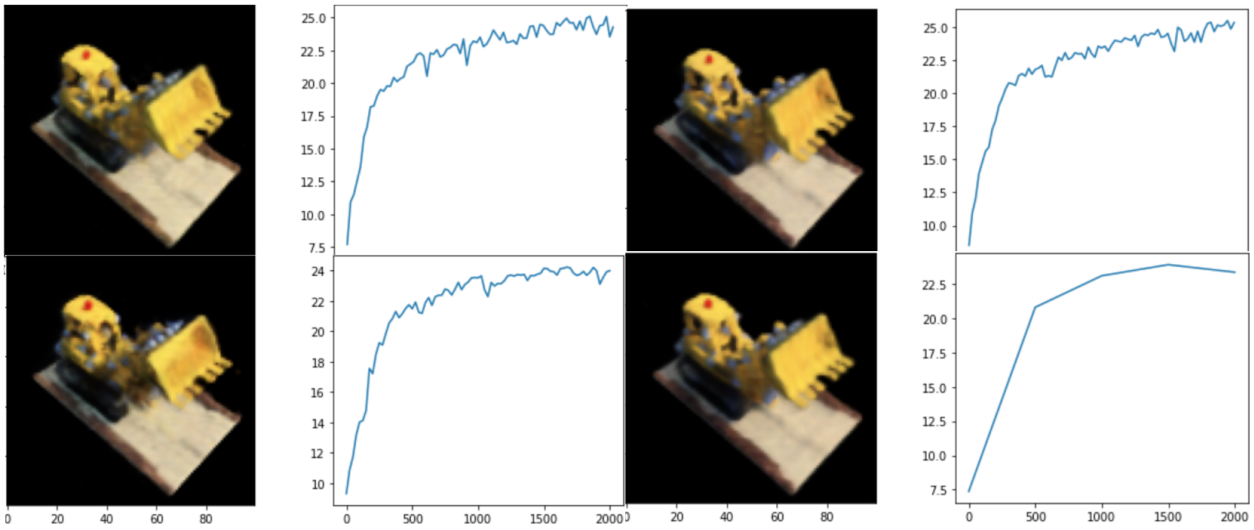

Figure 5. Visualization of rendered images using various number of training images. (Top Left) 5 images. (Bottom Left) 10 images. (Top Right) 25 images. (Bottom Right) 100 images.

4. Experiments

4.1. Data

Since the input for NeRF contains the viewing direction which is represented using camera poses. It requires a known camera matrix. Due to the fact that calibration procedure might be time-consuming and error-prone, for this project, we are using the tiny_nerf_dataset provided by NeRF. The dataset contains 106 different views of lego along with corresponding camera poses.

4.2. Result

The experiment results are listed in Table 5 shows some visualizations of rendered image using models trained with different size training set. We can have the following observations:

1. 2000 iterations is a reasonable value in terms of training progress. Considering the potential optimization techniques that can save at least 2x training time (e.g. JAX implementation), it's reasonable to expect an application that can generate a video within one minute. It should make the application practical to be used in daily use case.

2. Beyond iteration times, we can also observe that the more training images, the higher quality of the image especially within the area with high frequency details.

3. Surprisingly, the PSNR of the rendered image won't be downgraded a lot no matter what the size of the training set is. With only 5 images from different angles, we are able to generate a 360 video (https://github.com/zenXun/NeRF-Applications) which illustrates the power of NeRF.

5. Conclusion

By fine tuning the model, we find the best combination of hyper parameters of NeRF is (training size = 5, L = 10, N_c = 64), with which we are able to synthesize unseen view with a PSNR around 24. We can also observe that a high N_c works best with a large training set (experiment 1 & 3) and regresses the model with a smaller training set (experiment 7 & 8). Although the images are quite blurry without high frequency details, we show the potentials of using NeRF in practical applications. With other optimization techniques listed in Related Work section (e.g. meta-learning and JAX implementation), we can expect the model to output images of higher quality.

Detailed implementation can be found at https://github.com/zenXun/NeRF-Applications.

References

[1] F. Dellaert and L. Yen-Chen. Neural volume rendering: Nerf and beyond. *arXiv preprint arXiv:2101.05204*, 2020.

[2] B. Deng, J. T. Barron, and P. P. Srinivasan. JaxNeRF: an efficient JAX implementation of NeRF, 2020.

[3] L. Liu, J. Gu, K. Z. Lin, T.-S. Chua, and C. Theobalt. Neural sparse voxel fields. *arXiv preprint arXiv:2007.11571*, 2020.

[4] S. Lombardi, T. Simon, J. Saragih, G. Schwartz, A. Lehrmann, and Y. Sheikh. Neural volumes: Learning dynamic renderable volumes from images. *arXiv preprint arXiv:1906.07751*, 2019.

[5] B. Mildenhall, P. P. Srinivasan, R. Ortiz-Cayon, N. K. Kalantari, R. Ramamoorthi, R. Ng, and A. Kar. Local light field fusion: Practical view synthesis with prescriptive sampling guidelines. *ACM Transactions on Graphics (TOG)*, 38(4):1–14, 2019.

[6] 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. In *European Conference on Computer Vision*, pages 405–421. Springer, 2020.

[7] K. Park, U. Sinha, J. T. Barron, S. Bouaziz, D. B. Goldman, S. M. Seitz, and R.-M. Brualla. Deformable neural radiance fields. *arXiv preprint arXiv:2011.12948*, 2020.

[8] M. Tancik, B. Mildenhall, T. Wang, D. Schmidt, P. P. Srinivasan, J. T. Barron, and R. Ng. Learned initializations for optimizing coordinate-based neural representations. *arXiv preprint arXiv:2012.02189*, 2020.