CSE291 NeRF Questions

Sambaran Ghosal

December 2022

1 What is a radiance field? What information is included in a radiance field? How is neural radiance field (NeRF) different?

1.1 Answer

A radiance field is defined as a mapping $\mathcal{L} : R^3 \times S^2 \rightarrow R^3$, taking a point (x, y, z) and its corresponding viewing direction in the camera frame (θ, ϕ) to the color space defined by the RGB values at that position and direction. Basically radiance field contains the information of the RGB values at different points in the image space.

A neural radial field is a neural network approximation that takes in as input the point and directions of the image space and outputs the rgb value and color density function at each point.

2 What is ray marching? Given a radiance field, how is each pixel calculated? (This is called the render equation.) Write down your render equation in a concrete math expression with clarified notations.

2.1 Answer

Ray Marching is defined as the process of obtaining the rendered image from the neural radiance field output i.e. rendering the scene based on the rgb values of each position along the ray and the density function along the ray. The volume rendering assigns color to a particular position based on the probability that the ray can travel upto this point without hitting any other particle. We can sample points along the ray using stratified sampling as follows

$$t_i \sim \mathcal{U}[t_n + \frac{i-1}{N}(t_f - t_n), t_n + \frac{i}{N}(t_f - t_n)] \tag{1}$$

where t_n, t_f are the near and far bound planes of the ray, N is the number of samples we need and $i \in [1, 2, 3, ..., N]$. Let $c_i = (r, g, b)_i$ be the output color from the MLP for i^{th} point along a ray, σ_i be the density function along at this point. We then estimate the color of a position using the quadrature integral approximation as

$$\hat{C}(r) = \sum_{i=1}^{N} e^{-\sum_{j=1}^{i-1} \sigma_j \delta_j} (1 - e^{-\sigma_i \delta_i})) c_i \tag{2}$$

where $\delta_i = t_{i+1} - t_i$ is distance between adjacent samples along the ray and t_i are generated by the stratified sampling algorithm along each ray. Hence we can repeat this process for all points along a ray and get the corresponding color value at that point. Doing this for all rays and all positions will hence generate our image / novel scene.

3 What is positional encoding? What is the purpose of positional encoding in NeRF models? Train your model without positional encoding and compare the results. You need to show at least two pairs of examples.

3.1 Answer

Neural networks despite being called universal function approximators, are not so good at approximating high frequency / varying functions. Deep Learning methods tend to learn low frequency function approximators to the input data. Findings by Rahamanet al. [**?**] shows that mapping the inputs to first a higher dimensional space using frequency functions like sin, cos before feeding them to the neural networks helps better fit the data with high frequency variation.

If the MLP is represented as $_\theta$ as a mapping from $xyz\theta\phi$ to color space as $F_\theta : xyz\theta\phi \rightarrow c$, then the positional encoding can be defined as first passing the input $xyz\theta\phi$ to a frequency encoder $\gamma(p)$ which in our case is a harmonic embedding defined by

$$\gamma(p) = (sin(2^0 p), cos(2^0 p), sin(2^1 p), cos(2^1 p), ..., sin(2^{L-1} p), cos(2^{L-1} p) \tag{3}$$

where L is the number of frequency embeddings desired. The MLP is now defined as $F_\theta := F_\theta \; o \; \gamma(p)$ where the positional encoding is applied to each component of position and viewing direction vector.

The performance of two networks, one with positional encoder and one without positional encoder is shown below in Figure(**??**)

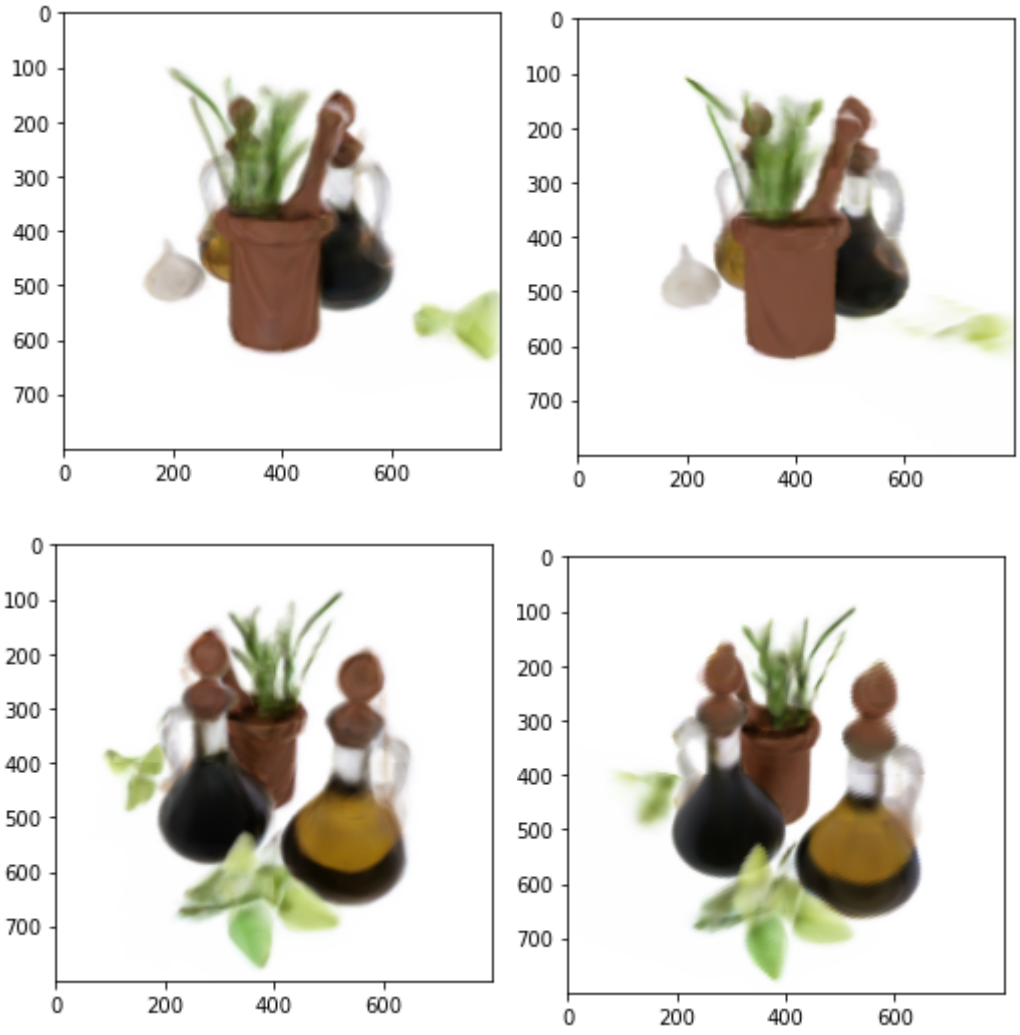

Figure 1: Generate image with no positional encoder (left) and with encoding (right)

4 Is it possible to extract scene geometry (e.g. depth) information from a trained NeRF model? Describe your method in detail. Implement your method and show two depth maps generated by your method.

4.1 answer

Yes, it is possible to produce depth map from a trained NeRF model. From NeRF, we have the sampled points along rays. Let $z_i = t_i - t_{i-1}$ be the distance between adjacent sampled points. From the NeRF MLP, we have the density function at each sampled point. Intuitively, density at each point somehow related to what is the probability that the ray travels upto a given point without being obstructed by other points. Mathematically, the term $e^{-\sum_{j=1}^{i-1}\sigma_j\delta_j}(1 - e^{-\sigma_i\delta_i})$) exactly represents the probability that the ray travels upto the i^{th} point in the ray without being obstructed i.e. lower this term is for a given point, higher chance that the ray terminates at this point. So if the term becomes 0 at a point, this indicates that the ray terminates at this point and hence the depth of this point in the image is a weighted sum of the distance between different samples along the ray weighted by the probability that the ray did not terminated at those points upto this point i.e. depth of a ray can be given by

$$d = \sum_{i=1}^{N} e^{-\sum_{j=1}^{i-1}\sigma_j\delta_j}(1 - e^{-\sigma_i\delta_i}))z_i \tag{4}$$

Computing this for each ray in the image hence gives us the depth map of the given scene.

Below in Figure(2) are attached two depth maps for two different scenes.

5 What are the major issues you find when using NeRF? List at least 2 drawbacks. For each of them, propose a possible improvement. You are encouraged to check follow-up papers of NeRF, but you should cite these works if borrowing their ideas.

5.1 Answer

- The rendering procedure used by neural radiance fields (NeRF) samples a scene with a single ray per pixel and may therefore produce renderings that are excessively blurred or aliased when training or testing images

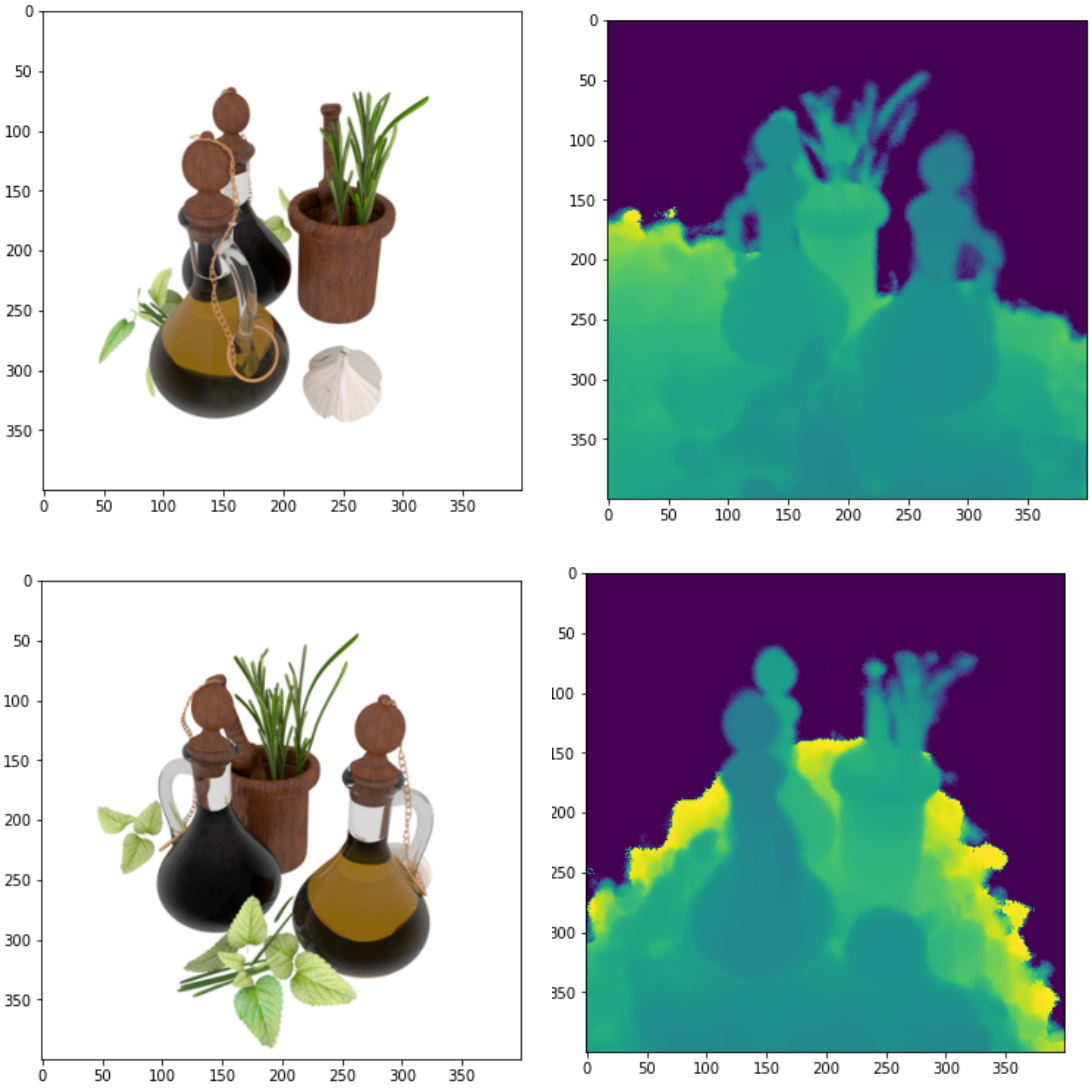

Figure 2: Ground truth and corresponding depth map of image scene

observe scene content at different resolutions. One may try to solve this problem by using multiple rays per pixel. But then we would have to query the MLP for the color space and density output for each ray and this increases the computational complexity a lot.

Solution Proposed Typical positional encoding as used in Neural Radiance Fields maps a single point in space to a feature vector, where each element is generated by a harmonic embedding with an exponentially increasing frequency.

Barron et al. [**?**] proposed their integrated positional encoding that considers Gaussian regions of space, rather than infinitesimal points to the positional encoder. This provides a natural way to input a "region" of space as query to a coordinate-based neural network, allowing the network to reason about sampling and aliasing. The expected value of each positional encoding component has a simple closed form:

$$E_{x\sim\mathcal{N}(\mu,\sigma^2)}\gamma_\omega(p) = sin(\omega\mu)e^{-\frac{(\omega\sigma)^2}{2}} \tag{5}$$

Rather than casting an infinitesimal ray through each pixel, they instead cast a full 3D cone. For each queried point along a ray, they consider its associated 3D conical frustum. Two different cameras viewing the same point in space may result in vastly different conical frustums. In order to pass this information through the NeRF network, they fit a multivariate Gaussian to the conical frustum and use the integrated positional encoding described above to create the input feature vector to the network.

This work is called **Mip-NeRF** and is described here : `https://jonbarron.info/mipnerf/`

- Optimizing a coordinate-based network from randomly initialized weights for each new signal is highly inefficient. The network weights θ are typically optimized via gradient descent to produce the desired image. However, finding good parameters can be computationally expensive, and the full optimization process must be repeated for each new target.

  Solution Tancik et al. [**?**] propose a meta-learning based weight initialisation that makes it easier to optimize the query network function weights. The simple-NeRF formulation relies on multi-view consistency for supervision and therefore fails if naively applied to the task of single view reconstruction. However, if the model is trained starting from meta-learned initial weights, it is able to recover 3D geometry. The MV Meta initialization has access to multiple views per object during meta-learning, whereas the SV Meta initialization only has access to a single view per object during meta-learning.

  This work is called **Learned Initializations for Optimizing Coordinate-Based Neural Representations** and is described here : `https://www.matthewtancik.com/learnit`

NeRF

Sambaran Ghosal
Department of Electrical and Computer Engineering
UC San Diego
La Jolla, USA
sghosal@ucsd.edu

I. Introduction

Neural Radiance Field or NeRF is a breakthrough work in the field of deep learning related to 3D data generation. It is a method of generating novel views from given different camera poses. It has wide applications since using novel views of scenes, we can create 3D meshes, create simulation scenes etc.

II. Problem Formulation

Given a set of camera poses in the world frame, the aim is to create 3D scene images. During training, we are provided with images corresponding to the scene observed from a given camera pose. We use this image and pose to train the NeRF model, and during generation, we then use the camera pose to generate rays for the scene and using NeRF output and volume rendering concepts, accumulate the rgb values of each point in the pixel space. This then yields the generated test scene. We will now discuss in detail the training architecture models.

III. Technical Approach

A. Training

1) ***Getting rays and origins****:* During training, NeRF requires the ray origin and directions corresponding to the given camera pose. Each point in the image is then described by the equation

$$r(t) = o + td \tag{1}$$

where $r(t)$ is the position of a pixel, o is the ray origin and d is the direction vector or viewing direction from the camera to the pixel. t is a parameter ranging from t_n corresponding to the nearest plane of image formation and t_f corresponding to the farthest plane. For our work, we use a Stratified Sampling approach to generate 64 points uniformly along each ray of the image. Hence from one image, we have in total $H \times W \times 64 \times 3$ points sampled from the image. The viewing direction is represented as another 1×3 vector at each pixel. Hence we have a total of $H \times W \times 3$ matrix as the direction vector for each pixel.

2) ***Positional Encoding****:* Once we have the points sampled along each ray and corresponding directions, we pass the points and the direction vectors through a positional encoder that maps these lower dimensional vectors to a higher dimensional vector using harmonic embedding. For a given input x, the harmonic embedding is defined as

$$E(x) = \begin{bmatrix} x \\ sin(x) \\ cos(x) \\ sin(2x) \\ cos(2x) \\ .. \\ sin(2^{N-1}x) \\ cos(2^{N-1}x \end{bmatrix} \tag{2}$$

where N is the desired number of encoding dimension. So for each component of point (x, y, z) and each component of direction vector (d_1, d_2, d_3) we apply the encoding. If the number of positional encoding for position is N_{pos} and encoding dimension for direction is N_{dir}, the output dimension after encoding will be $3 \times (1 + 2N_{pos})$ for position and $3 \times (1 + 2N_{dir})$ for direction.

3) ***Neural Radiance Field Model Architecture****:* The main concept of NeRF is that a scene can be modeled as a collection of rgb values and density function at each point in the camera space. We can get these rgb values and the density values at each point in the space by using a non-Convolutional Neural Network (MultiLayer Perceptron) that takes in as input the encoded / or non encoded position and viewing direction at each point in the image space. The architecture used in our work is as follows :-

- We use 4 linear layers with 128 neurons at each layer. These layers take in as input the position of the points. We have a skip connection in the 2nd layer where we concatenate the output of this layer with the original position input.
- The output of the first 4 layers is then processed by two separate Linear Layers, one producing the density function, and the other producing a temporary output which is further processed as discussed below.
- The intermediate output is then concatenated with the viewing direction of the points, and then further processed by two separate linear layers which finally outputs the rgb values at the points. The rgb values and the density output are finally concatenated to give the final output of the NeRF MLP which will now be used first with volume rendering to generate scenes and then use the ground truth

images provided corresponding to different camera poses to train this model parameters.
The architecture flow is shown in Figure(2).

*4) **Volume Rendering**:* Once we have the rgb values and the density function value at each point, we use volume rendering concepts to generate the image scene. Let $c_i = (r, g, b)_i$ be the output color from the MLP for points along ray i, σ_i be the density function along the ray i. Then the obtained rgb values along ray is given as

$$\hat{C}(r) = \sum_{i=1}^{N} e^{-\sum_{j=1}^{i-1} \sigma_j \delta_j} (1 - e^{-\sigma_i \delta_i}) c_i \qquad (3)$$

where $\delta_i = t_{i+1} - t_i$ is distance between adjacent samples along the ray and t_i are generated by the stratified sampling algorithm along each ray. This equation accumulates rgb values along the ray depending on the density function which represents the chance that a given pixel is actually occluded.
For our work due to computation resource limitation and time limitation, we did not run the training using Fine Model that uses Hierarchial Sampling to even further generate points that are not occluded and hence have more impact in generating the scenes.
Hence, the overall training architecture flow is described in Figure(1).

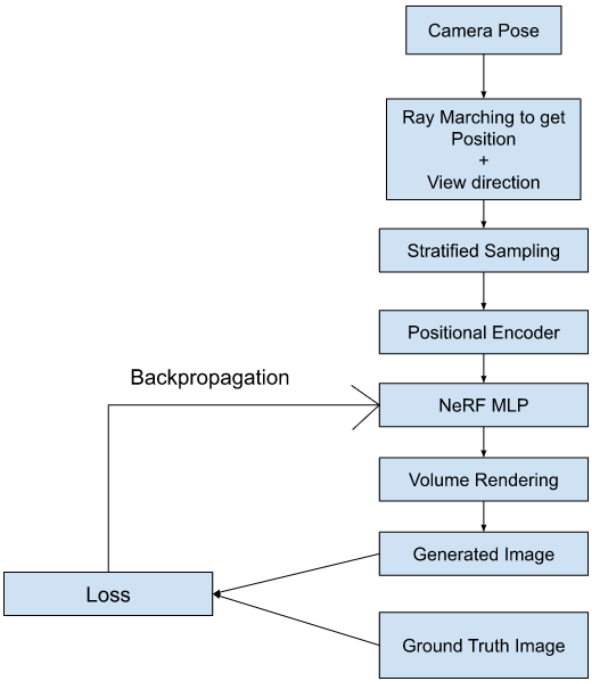

Fig. 1: Nerf pipeline

IV. Experiments

A. Train setting

During training, we use the following hyperparameter values : Learning rate is started from 4e-3 and decayed by 0.25 after every 50000 iterations. We only use coarse network and do not use fine network and no hierarchial sampling. The positional encoding dimensions are set at 5 for positions and 2 for viewing directions. The MLP has 4 layers with skip connection after second layer in addition to the bottleneck layers that are not changed anytime. The 4 linear layers are set with dimension 128. The network is trained on 100×100 sized images and the first 100 images for the training + validation set provied to us. Evaluation of the model is done every 500 iterations on an image not included in the training set and the progress is saved. We keep track of training and validation PSNR to evaluate the model performance.

Since NeRF has a high GPU requirement, we use a batch size of 4096 to select rays from a given image. This helps reduce the GPU usage slightly.

B. Training Progress

Figure (3) show the quality of performance over several iterations. The left hand side shows the generated image using NeRF on a validation pose, middle one is the ground truth image for that validation file and right side image shows the training psnr vs validation psnr.

C. Training with and without Positional Encoding

In this section, we study the affect of doing training with and without positional encoding on the position and directions. For the position encoding we use the same dimensions as above i.e. position encoding of 5 for (x, y, z) and directional encoding of 2. We show the 800×800 image reconstructed using model trained with and without encoding. We also report the PSNR on the validation image using the above.

The models are tested on validation images 1_val_0050.png and 1_val_0080.png. The resulting generated images are shown in Figure(4). The model with no encoder yield a PSNR of 21.4405 on the 1_val_0050.png and 20.0150 on 1_val_0080.png, whereas the model with the encoder yields a PSNR of 21.7359 on the 1_val_0050.png and 21.4636 on 1_val_0080.png.

The differences would be more clear in the image if we had also used the hierarchial sampling and fine model output too. But we can observe in the left figure, we can see that the grass in the right hand side of the upper left image is represented better than in the right image showing that with positional encoding we are able to capture different objects. Similar behaviour is also observed in the bottom image where grasses in the surface and tub are better represented in the left image than in the right image.

D. Validation PSNR

Once the training is done, we use the NeRF model to generate 800×800 sized images on poses labeled '1_val_0020.txt', '1_val_0040.txt', '1_val_0050.txt', '1_val_0095.txt' that were

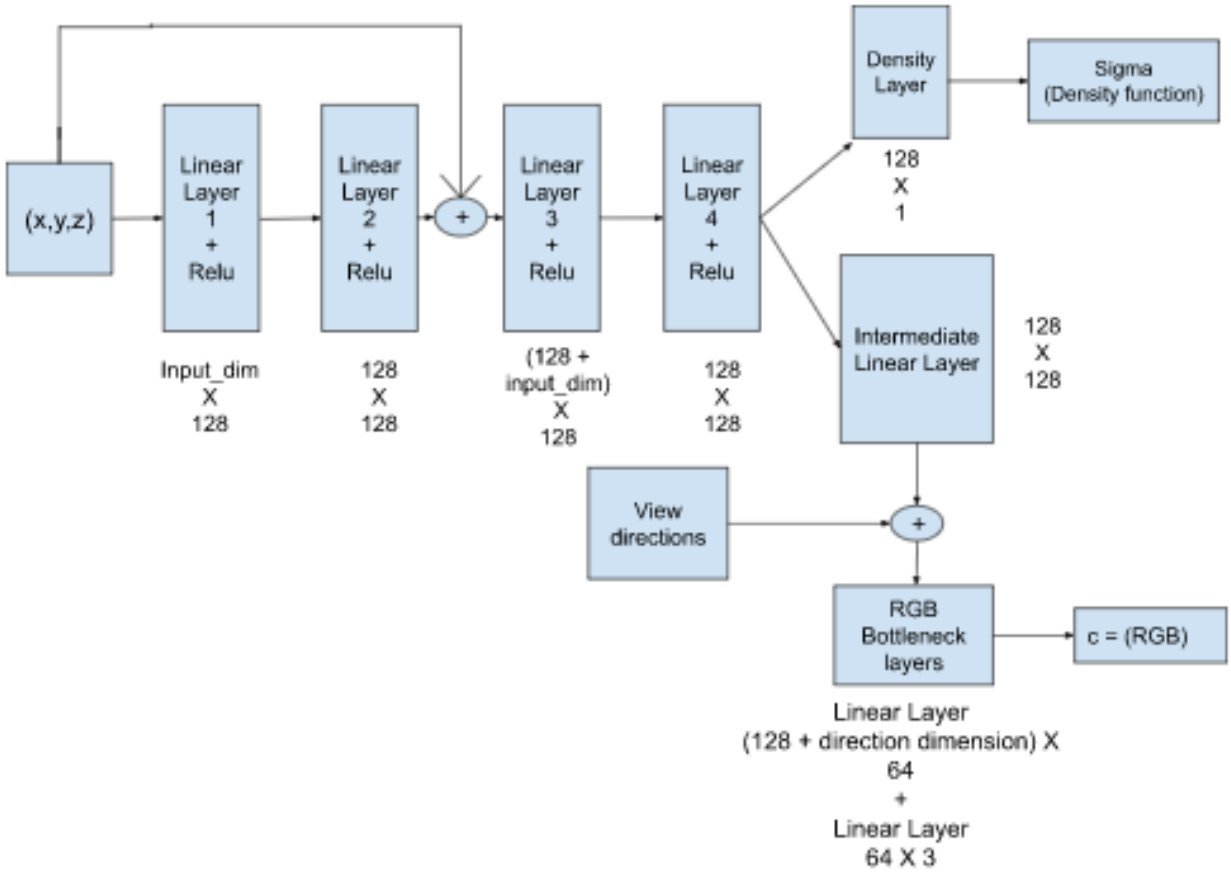

Fig. 2: NeRF MLP Architecture

Validation Image	PSNR
1_val_0020.png	23.84
1_val_0040.png	22
1_val_0050.png	21.12
1_val_0095.png	22.05

TABLE I: PSNR of selected validation images

not part of the training image and the validation image during training. The generated images vs ground truth images are shown in Figure(5) and corresponding PSNR is shown in Table(I).

Using our trained NeRF, we are able to achieve an average PSNR of **22.06**. We get a low PSNR which can be credited due to the fact that we used only a linear layer of 128 feature dimensions, a total of 4 linear layers, a lower encoding dimension of 5 and 2, and also not using the fine model for further quality improvement. We can see that our generated images although having very similar scenes, are blurry compared to the sharp quality of the ground truth images.

E. Test Settings

To generate the test images, we simply use the given pose of the camera to get all rays and directions. Then we generate the total 800×800 image by generation 4 blocks of 400×400 images corresponding to upper left, upper right, bottom left and bottom right blocks of the image and concatenate these accordingly to get the full image. This way at a given instant we are using only 1/4 GPU memory compared to if we did generate the image at once which would quite possibly lead to GPU OOM issues.

F. Test image generation

Finally, using the trained model, we generate novel views for the camera poses provided in 2_test_0000 , 2_test_0016 , 2_test_0055 , 2_test_0093 , and 2_test_0160. These test images are attached in the zip file along with the submission.

V. Acknowledgement

I have written the code by referring to available online NeRF implementations from [1], [2] and [3]. I acknowledge that I have not copied the full code directly and have understood

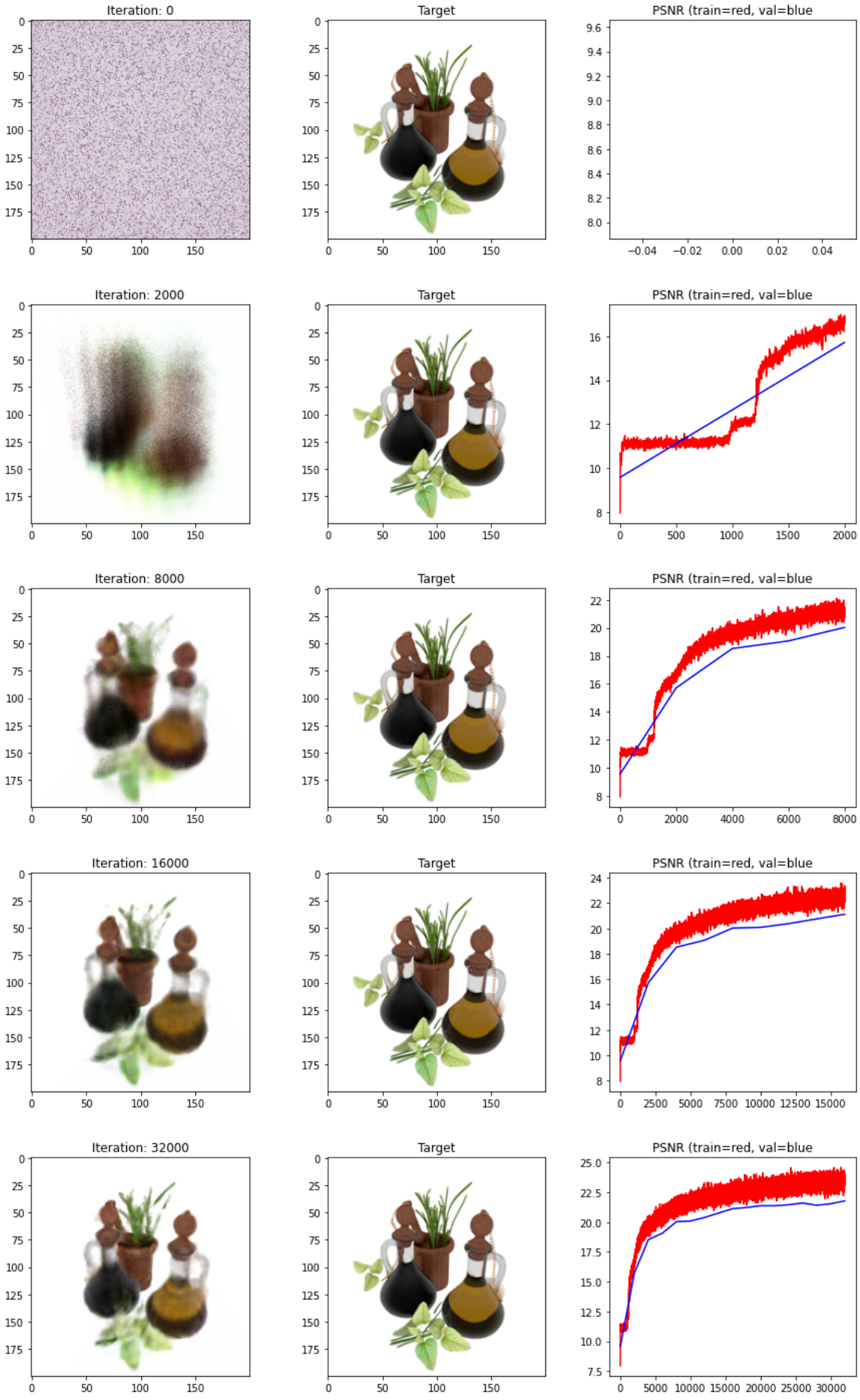

Fig. 3: Validation image progress over epochs

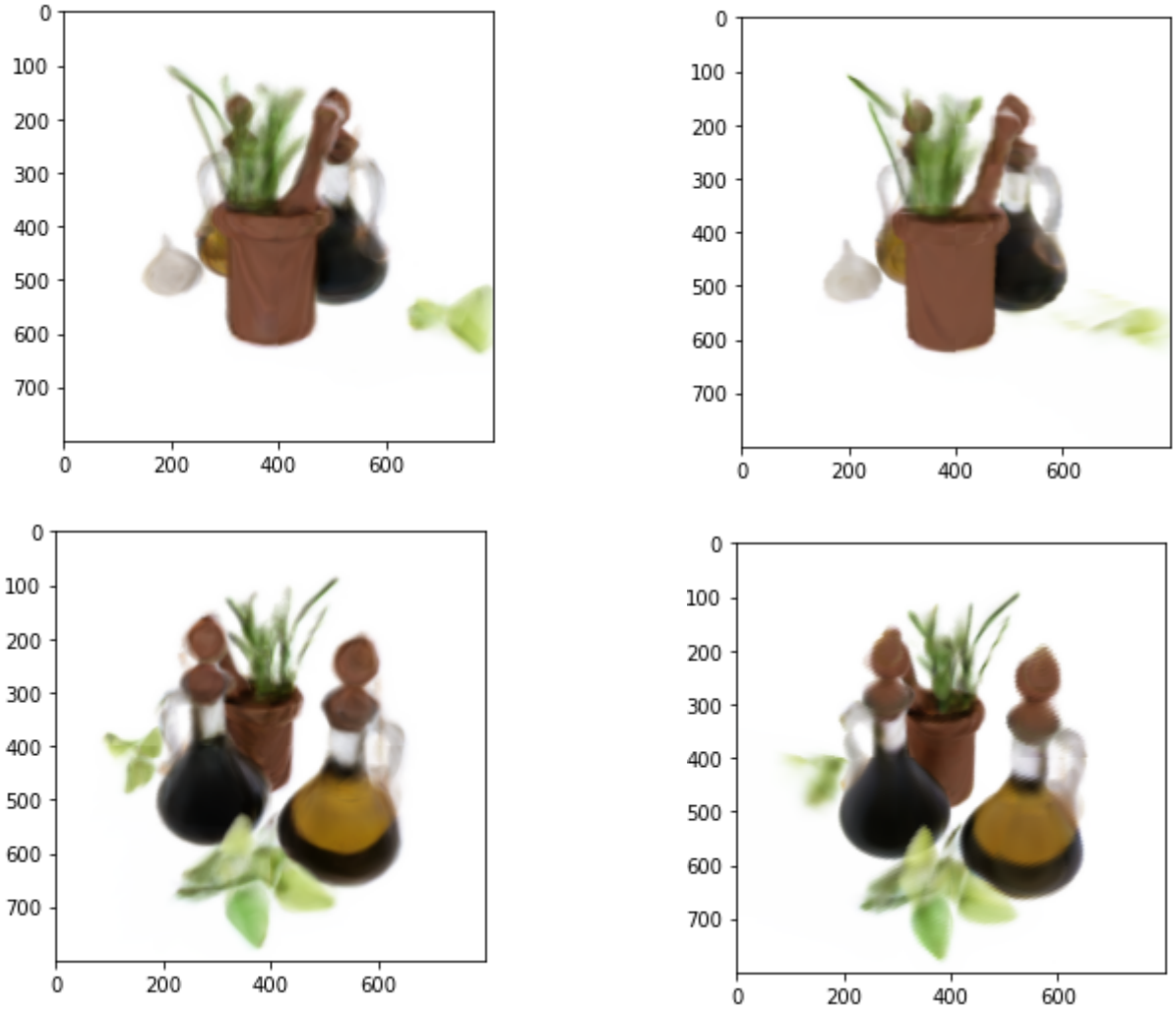

Fig. 4: Generate image with positional encoder (left) and without encoding (right)

each step in the given implementations and written down the implementations in my own way.

I would also like to acknowledge that I had fruitful discussion and debugging sessions with my colleague Mr Albert Liao. We discussed key implementation features, concepts and also helped each other with pytorch debugging sessions.

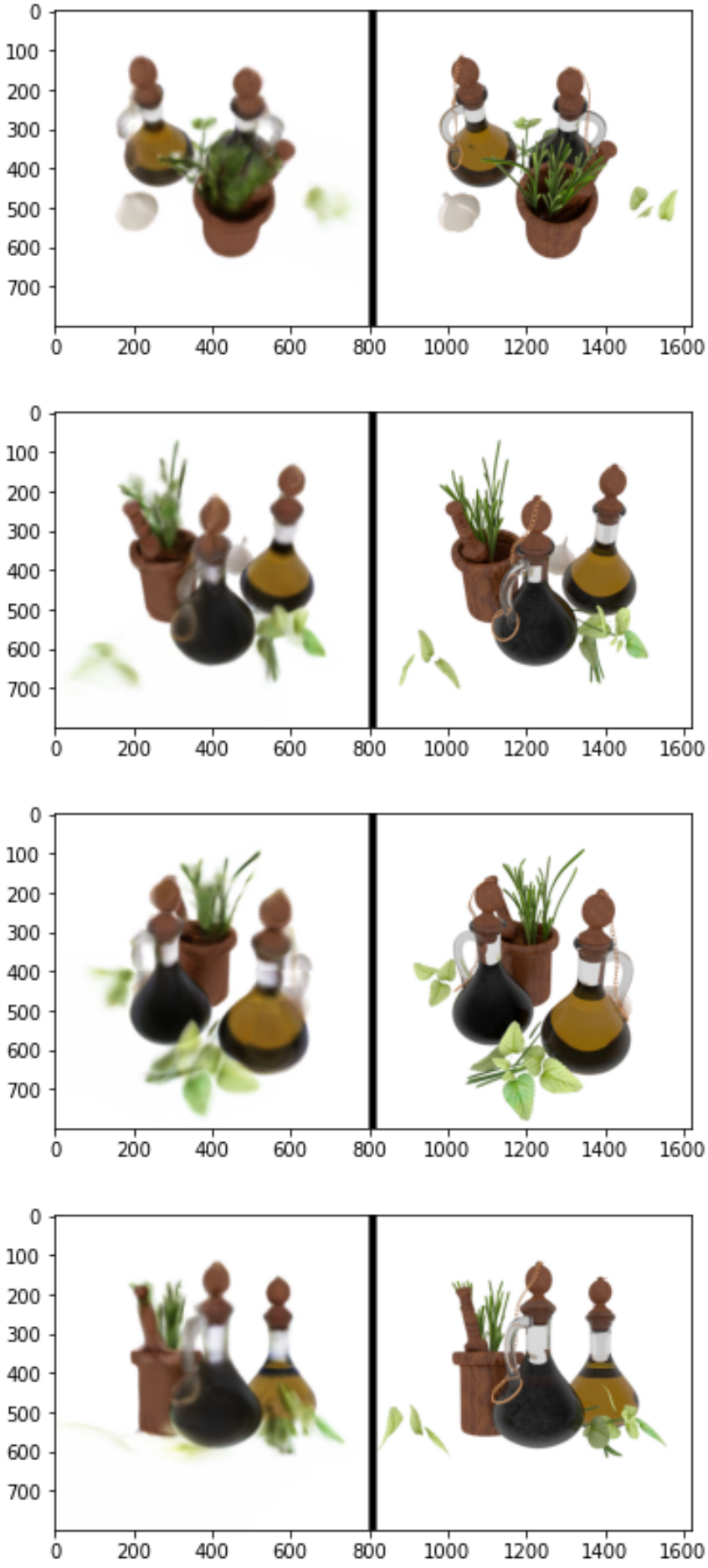

Fig. 5: Generated images using NeRF(left) and ground truth images (right)

REFERENCES

[1] "'Mildenhall et al. NeRF: Representing Scenes as Neural Radiance Fields for View, https://github.com/bmild/nerf Synthesis"'

[2] "'https://towardsdatascience.com/its-nerf-from-nothing-build-a-vanilla-nerf-with-pytorch-7846e4c45666"'

[3] "'https://github.com/yenchenlin/nerf-pytorch"'