CS 499 - Mid-semester Project Report Submission

[Aditi Dey - 20110007, Kareena Beniwal - 20110095]

NeRF - (Neural Radiance Fields)

Problem Statement:

What if there was a way to capture the entire 3D scene just from a sparse set of 2D pictures?

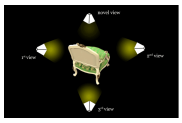


Fig1: captured 2D pictures & novel view

Reviewed Literature:

https://arxiv.org/abs/2003.08934

Theory:

1. The input to the model is represented by a 5D vector-valued function consisting of 3D location ***x*** = (x, y, z) and 2D viewing direction (θ, φ), and the output is an emitted color c = (r, g, b) and volume density σ.
2. The volume density σ as a function of only the location ***x*** , while the RGB color c is predicted as a function of both location and viewing direction.
3. We start by applying the camera to the world matrix (transform_matrix in the json file). The elements denoted by r are used for rotation and the elements denoted by t are used for translation. The last row is required to make the matrix homogeneous. This is the *extrinsic transformation matrix*. This matrix gives us the x, y, z and φ input variables.

$$C_{ex} = \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

4. We project the rays (given by rays_origin and rays_direction in the nerf_components.py file) received from the object on the image plane using the *intrinsic transformation matrix* (because it uses factors such as focal length that are internal properties of the camera)

$$C_{in} = \begin{bmatrix} f & 0 & o_x & 0 \\ 0 & f & o_y & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

5. Next, we apply *positional encoding* and *Hierarchical volume sampling* on the emitted rays. We first sample a set of locations on the rays using uniform or stratified sampling, and evaluate the coarse network at these locations by using inverse transform sampling, and then perform fine sampling on some selected locations depending on the density.
6. Next we render the rgb and density values at the finely sampled points to generate the output images.

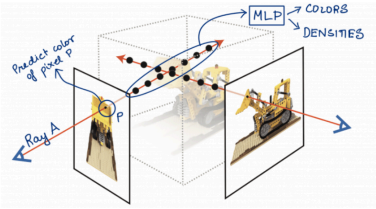


Fig2: Volumetric Rendering Schematic

Dataset:

We have used the nerf synthetic dataset and trained it for the lego image. There are 100 images, capturing the lego object from different values of (x, y, z, θ, φ).
The images have also been processed into a json file, which stores rotation angle and transformation matrix for each frame.

Code Brief:

https://github.com/AditiDey29/NeRF

The code comprises of the following files:

Files	Functionality
dataloader.py	This file defines a class NerfDataLoader, which stores the camera to world matrix, focal length, and direction vectors.
evaluate.py	Evaluates the pre-trained NeRF model on a validation dataset, computing and logging metrics such as MSE, PSNR, SSIM, LPIPS for each image.
nerf_components .py	Contains functions for ray-sampling, encoding positions, and rendering rgb-depth images using NeRF
nerf_model.py	Defines NN architecture - NerfNet, which takes input rays, and (θ, φ) to predict the output. Network contains layers with skip connections.
render_view.py	Render novel views by predicting rgb and depth info for each pixel.
train.py	We follow the procedure mentioned in theory using various functions defined in nerf_components.py and save the generated output images.

Training:

The trained model results in two major outputs:
1. Depth Maps: represents the estimated depth values for each pixel in the scene i.e. provides information about the distance from the camera to objects in the scene.
2. Reconstructed image: represents the synthesized view of the scene.
Depth maps and the images are generated at certain intervals during the training for a randomly selected subset of views.

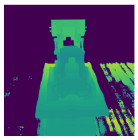
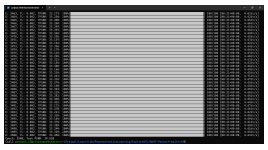



Fig3: Depth map & the corresponding reconstructed image

Training Results:



Total no of epochs: 3500, Test PSNR: 39.520

Near-future Advancements:

1. *Focusing on Computational Intensity*: Training took 24 hrs to complete, therefore we are planning to transform the code using PyTorch lightning and also make use of pytorch quantization.
2. We will also be studying TensoRF(memory efficient as compared to NeRF) and Nerf Codebooks.