NeRF Project

Neural Radiance Fields (NeRF)

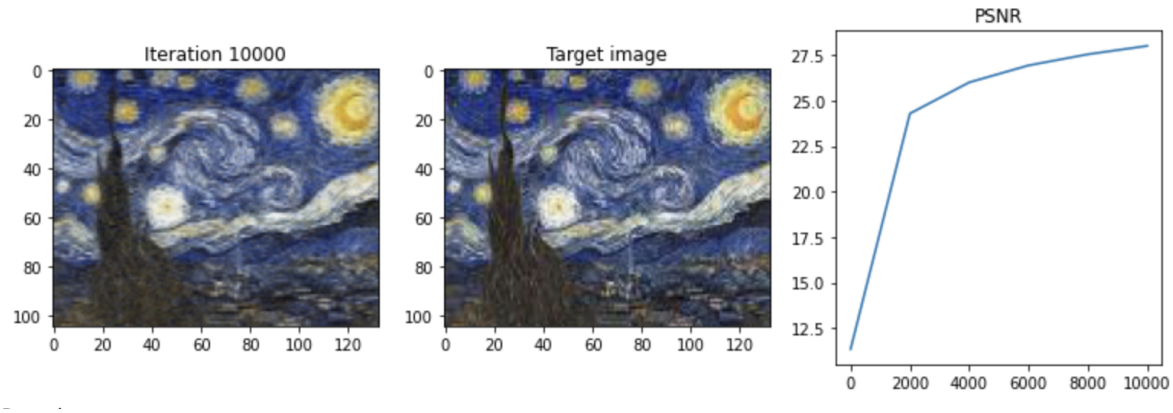
Motivation

The Neural Radiance Field, or NeRF, is a fairly new paradigm in the world of deep learning and computer vision. Introduced in the ECCV 2020 paper “NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis” (which received an honorable mention for Best Paper), the technique has since exploded in popularity and has received more than 1500 citations thus far[1]. The method marks a dramatic change from the conventional ways by which machine learning handles 3D data. The visuals certainly have that “wow” factor as well, which you can see on the project page. In this project, we aim to implement the basics of NeRF and explore interesting applications that build upon this paradigm.

Tasks

There are several pieces we ask you to implement in this project:

1) Fit a single image with positional encoding



1. Standard coordinate-based MLPs cannot natively represent high-frequency functions on low-dimensional domains. To better preserve high-frequency variation in the data, we can encode each of the scalar input coordinates with a sequence of sinusoids with exponentially increasing frequencies before passing them into the network.

   More specifically, g given a coordinate x, we map each coordinate value p in x with the encoding function $\gamma(\cdot)$ as below:

   $\gamma(p) = (sin(2^0\pi p), cos(2^0\pi p), sin(2^1\pi p), cos(2^1\pi p), ..., sin(2^{L-1}\pi p), cos(2^{L-1}\pi p))$where the dimension of the output $\gamma(p)$ is $2L$.

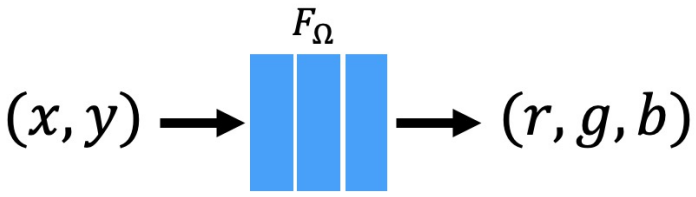
   If the input is a 2D coordinate $x = (p_1, p_2)$, after positional encoding we will get a long vector with dimension $4L$:

   $(sin(2^0\pi p_1), sin(2^0\pi p_2), cos(2^0\pi p_1), cos(2^0\pi p_2), sin(2^1\pi p_1), sin(2^1\pi p_2), cos(2^1\pi p_1), cos(2^1\pi p_2), ..., sin(2^{L-1}\pi p_1), sin(2^{L-1}\pi p_2), cos(2^{L-1}\pi p_1), cos(2^{L-1}\pi p_2))$

   Similarly, you can encode a 3D coordinate or an input signal of any dimension.

2. 2D Image Fitting.

   Now, let's try to fit a 2D image with a multilayer perceptron (MLP). We learn that we can store a 2D image with a coordinate-based MLP (as shown in the figure below). The input to this MLP is 2D pixel coordinate (x, y) as a pair of floating point numbers, and the output is RGB color of the corresponding pixel. This is a simple supervised learning problem, and we can just use simple gradient descent to train the network weights and see what happens.

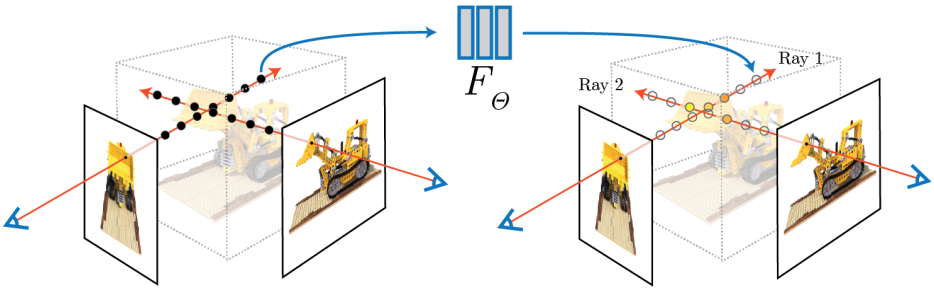


   You need to define the network architecture for this 2D fitting task and report the PSNR metric after training is converged. You should try different model structures (e.g. w/wo positional embedding, number of frequency, more/fewer layers, smaller/bigger hidden dimensions) and give analysis.

   - PSNR is an image quality measurement. Higher PSNR generally indicates that the reconstruction is of higher quality.

2) Implement NeRF and fit on multi-view images

A simplified version of NeRF represents a continuous scene as a function using the following MLP network, whose input is a 3D location $x = (x, y, z)$ and whose output is an RGB color $c = (r, g, b)$ and volume density σ at that 3D location.



We can render an image from this neural radiance field by estimating the color at each pixel in the image by shooting a camera ray from that pixel through the scene, and accumulating color and density along the way.

Rendering an image from neural radiance fields consists of the following steps:

1. Given camera parameters, compute the origin and the direction of the camera ray through each pixel of the image in the world coordinate frame.
   a. Get all the pixel coordinates in the image space (shape: `[height, width, 2]`). Each 2D coordinate is (x, y), where x is the index along the horizontal axis, and y is the index along the vertical axis.
   b. Transform image coordinates into camera coordinates using the camera intrinsics K. You should first turn each pixel coordinate (x, y) into homogeneous coordinates $(x, y, 1)$ and then apply K in some way.

Remember that K transforms camera coordinates into image coordinates, while here you are supposed to do the opposite.

c. In fact, the coordinates you computed in the last step are exactly the directions of each ray in the camera coordinate space because they can be viewed as a vector starting from the camera origin (0, 0, 0). Your task is to convert both the camera origin and the direction of each ray to the world coordinate frame using the camera-to-world transformation matrix.

2. Sample points along each camera ray and compute the sampled 3D points' coordinates.

a. Given the origin o and the direction d of a ray, we want to sample 3D points x on this ray. The sampling range is bounded by the near and far threshold, denoted as t_n, t_f. In order to ensure the ray to be thoroughly sampled during training, we evenly partition the ray into N bins and uniformly sample one point in each bin.

$x_i = o + t_i d, t_i \sim U(t_n + \frac{i-1}{N}(t_f - t_n), t_n + \frac{i}{N}(t_f - t_n))$

where $t_i, i \in \{1, 2, ..., N\}$ is the sampled depth value.

3. Composite the colors and densities of the sampled points along a camera ray to get the final color of the pixel.

a. The expected color C($\mathbf{r}$) of camera ray $\mathbf{r}$ in the continuous space can be computed with the formula:

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

where $T(t) = exp(-\int_{t_n}^{t} \sigma(r(s))ds)$ denotes the accumulated transmittance along the ray from t_n to t, i.e., the probability that the ray doesn't hit any other particle from t_n to t.

b. To numerically and differentiably estimate this continuous integral, we use a discrete set of samples with the quadrature rule as follows:

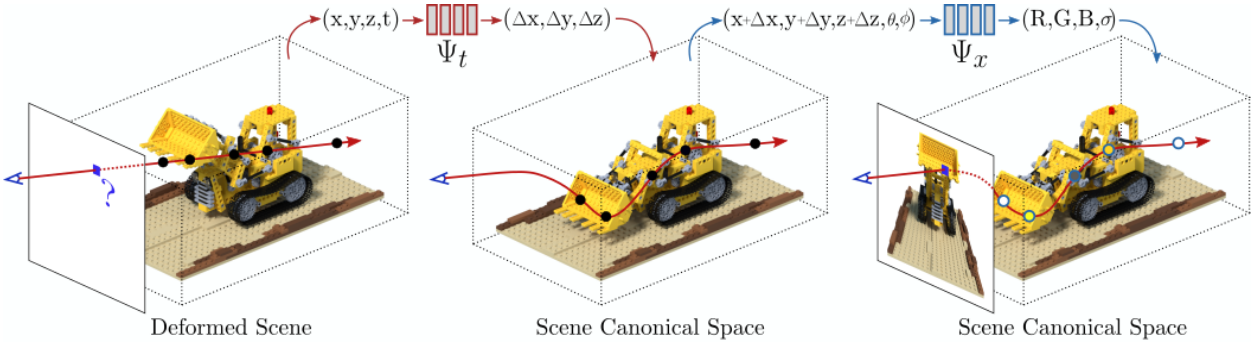
$C(r) = \sum_{i=1}^{N} T_i(1 - exp(-\sigma_i\delta_i))c_i$, where $T_i = exp(-\sum_{j=1}^{i-1} \sigma_j\delta_j)$ and $\delta_i = t_{i+1} - t_i$ is the distance between adjacent samples. From sampled depth values $t_i, i = 1, 2, ..., N$, you can get distances $\delta_i, i = 1, 2, ..., N-1$, and you should append a huge number $1e9$ as the last distance δ_N .

4. Render an image with trained NeRF.

Compute the pixel color by integrating colors of all samples along a ray, i.e. weighted sum of the color value according to the compositing weight $C(r) = \sum_{i=1}^{N} w_i c_i$. Repeating this operation for all pixels will allow us to render a final RGB image. To render a depth map instead, simply replace colors to sampled depth values, i.e., $D(r) = \sum_{i=1}^{N} w_i t_i$

5. Create a 360 video with the trained NeRF by rendering a set of images around the object. Evaluate the novel view synthesis results qualitatively. Evaluate the novel view synthesis results quantitatively on the test set with PSNR metric. Report the analysis of the model architecture/training strategy/positional encoding/…, etc.

3) NeRF extension: NeRF for dynamic scenes



NeRF is originally proposed to synthesize novel views for static scenes, where the same spatial location can be queried from different images. Here we explore how to extend neural radiance fields to a dynamic domain, allowing us to reconstruct and render novel images of objects under rigid and non-rigid motions from a single camera moving around the scene.

1. The original NeRF represents the scene using a 5D input of the query 3D location and viewing direction (x, y, z, θ, ϕ). The first idea to model dynamic scenes is a straight-forward extension of NeRF in which the scene is represented by a 6D input of the query 3D location, viewing direction and time $(x, y, z, t, \theta, \phi)$.
2. Another idea is to model the dynamic scene with two neural network modules Ψ_t, Ψ_x. On the one hand, we have the Canonical Network, an MLP $\Psi_x(x, d) \rightarrow (c, \sigma)$ is trained to encode the scene in the canonical configuration such that given a 3D point x and a view direction d returns its emitted color c and volume density σ. The second module is called Deformation Network and consists of another MLP $\Psi_t(x, t) \rightarrow \Delta x$, which predicts a deformation field defining the transformation between the scene at time t and the scene in its canonical configuration, as can be shown from the above figure.
3. Explore and implement new ideas to model the dynamic scene. This might include but not limited to new representation, auxiliary loss function, data augmentation, new embedding, etc.

4) Bonus: Create your own Nerf

Create your own Nerf with new techniques or novel applications based on your implementations. New techniques mean ideas to improve the existing Nerf methods. Novel applications indicate applying Nerf methods to a domain that no one has explored and is also meaningful/helpful.

Below we show an example of extending Nerf for decoupling the monocular video. Any invention (not been explored by anyone before) based on current Nerf is acceptable.

Example for your own Nerf: Decoupling dynamic and static objects from a monocular video [6]

Given a monocular video, segmenting and decoupling dynamic objects while recovering the static environment is a widely studied problem in machine intelligence. Based on Part 2) and Part 3), we can try to extend NeRF to a self-supervised approach that takes a monocular video and learns a 3D scene representation that decouples moving objects from the static background.

Grading

1. Fit a single image with positional encoding (20 pts)
    a. Implement the positional encoding function that can take input signal of any dimension(2D/3D/…)
    b. Implement the model architecture, training script for 2D image fitting
    c. Test different model structures (e.g. w/wo positional embedding, number of frequency, more/fewer layers, smaller/bigger hidden dimensions), report qualitative and quantitative results, and give analysis.
2. Implement NeRF and fit on multi-view images (30 pts)
    a. Implement the vanilla NeRF including the origin and camera ray direction computation, points sampling along each camera ray and volume rendering.
    b. Evaluate the novel view synthesis results qualitatively and quantitatively. Create a 360 video with the trained NeRF by rendering a set of images around the object. Report the analysis of the model architecture/training strategy/positional encoding/…, etc.
3. NeRF extension: NeRF for dynamic scenes (30 pts)

    Based on your codebase in Part 2, first, try to implement the two ideas of representing dynamic scenes. Evaluate the novel view synthesis on both time and space conditions. Give qualitative and quantitative results and analysis.
4. Propose new ideas for dynamic nerf (20pts)

    Based on 3, propose new ways to represent the dynamic scene or improve performance. Compare with the performance with 3 and give analyses.
5. Bonus: Create your own Nerf (20pts)

References

[1] Ben Mildenhall, Pratul P. Srinivasan, Matthew Tancik, Jonathan T. Barron, Ravi Ramamoorthi, Ren Ng — NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis (2020), ECCV 2020

[2] Zhengqi Li, Simon Niklaus, Noah Snavely, Oliver Wang — Neural Scene Flow Fields for Space-Time View Synthesis of Dynamic Scenes (2021), CVPR 2021

[3] Albert Pumarola, Enric Corona, Gerard Pons-Moll, Francesc Moreno-Noguer — D-NeRF: Neural Radiance Fields for Dynamic Scenes (2021), CVPR 2021

[4] Michael Niemeyer, Andreas Geiger — GIRAFFE: Representing Scenes as Compositional Generative Neural Feature Fields (2021), CVPR 2021

[5] Park, K., Sinha, U., Hedman, P., Barron, J. T., Bouaziz, S., Goldman, D. B., ... & Seitz, S. M. (2021). Hypernerf: A higher-dimensional representation for topologically varying neural radiance fields. *arXiv preprint arXiv:2106.13228*

[6] Wu, T., Zhong, F., Tagliasacchi, A., Cole, F., & Oztireli, C. (2022). D^2 NeRF: Self-Supervised Decoupling of Dynamic and Static Objects from a Monocular Video. *arXiv preprint arXiv:2205.15838*

[7] Alex Yu, Sara Fridovich-Keil, Matthew Tancik, Qinhong Chen, Benjamin Recht, Angjoo Kanazawa — Plenoxels: Radiance Fields without Neural Networks (2022), CVPR 2022 (Oral)