Project 6: Neural Radiance Fields (NeRF)

CS 6476

Spring 2025

Brief

- **Due:** Check Canvas for up-to-date information.
- Project materials are available on Canvas.
- Hand-in: through Gradescope.
- **IMPORTANT:** When submitting your report, *you must assign a slide to every problem even in the case of problems you do not complete, including extra credit.* The only exception is for "Bells & Whistles (alternate extra credit)," where it is acceptable to not attach a slide only if you do not try any alternate extra credit.
- **Required files:** `<your_gt_username>.zip`, `<your_gt_username>_proj6.pdf`.

Overview

In this project, you will design and train a neural radiance field (NeRF) network for arbitrary scene rendering. NeRFs were a groundbreaking approach that encoded the information about a 3D scene into a Multilayer Perceptron (a series of fully connected layers). For the best background on this topic, please consider reading through the original paper by Mildenhall et al. [2] and the companion video. It walks through how NeRFs function, the rationale behind the design decisions, and how they compared against alternative methods. This class's lecture on NeRFs is also a great resource on better understanding NeRFs and the more recent methods that have outperformed them. The Computerphile video on this topic is also good and includes an example of the output from a NeRF trained from basic videos of a real scene.

Setup

1. Follow Project 6 README.
2. Run the notebook using `jupyter notebook proj6_local.ipynb` inside the repository folder.
3. After implementing all functions, ensure that all sanity checks are passing by running `pytest tests` inside the repository folder.
4. If your laptop has a GPU or Mac has Apple Silicon chips, all training procedures can be done locally, which should take less than 10 minutes.
5. [**Follow below steps only if you need to train NeRF model using Google Colab**] After passing the unit tests, run `python zip_for_colab.py` which will create `cv_proj6_colab.zip`.
6. Upload `proj6_colab.ipynb` to Colab. This notebook can also be run locally, but it will be very slow without a CUDA-enabled NVIDIA GPU or Apple Silicon GPUs.
7. Upload `cv_proj6_colab.zip` to session storage.
8. Click `Run All` inside `Runtime` which will train your model.

Writeup

For this project (and all other projects), you must do a project report using the template slides provided to you. **Do not** change the order of the slides or remove any slides, as this will affect the grading process on Gradescope and you will be deducted points. **You must assign a slide to every problem even in the case of problems you do not complete, including extra credit.** The only exception is for “Bells & Whistles (alternate extra credit)”.

In the report you will describe your algorithm and any decisions you made to write your algorithm a particular way. Then you will show and discuss the results of your algorithm. The template slides provide guidance for what you should include in your report. A good writeup doesn’t just show results—it tries to draw some conclusions from the experiments. You must convert the slide deck into a PDF for your submission, and then assign each PDF page to the relevant question number on Gradescope.

If you choose to do anything extra, add slides **after the slides given in the template deck** to describe your implementation, results, and analysis. You will not receive full credit for your extra credit implementations if they are not described adequately in your writeup.

1 2D image representation

We aim to explore the task of representing a 2D image using a neural network. Specifically, we encode 2D coordinates using positional encoding and fit the image using a multilayer perceptron (MLP). The final goal is to enhance the performance of the MLP such that the Peak Signal-to-Noise Ratio (PSNR) of the reconstructed image exceeds 30 after training for 10,000 iterations. PSNR serves as a quantitative measure of image reconstruction quality, where higher values signify better reconstruction fidelity.

1.1 Positional Encoding

ReLU-activated neural networks learning directly from coordinates suffer from a *spectral bias*, where they have a hard time learning representations for high-frequency features. We can remedy this through the use of a positional encoding $\gamma(\mathbf{x}) : \mathbb{R}^n \to \mathbb{R}^{2Ln}$ as defined below:

$$\gamma(\mathbf{x}) = \begin{pmatrix} \sin(2^0\pi\mathbf{x}) \\ \cos(2^0\pi\mathbf{x}) \\ \vdots \\ \sin(2^{L-1}\pi\mathbf{x}) \\ \cos(2^{L-1}\pi\mathbf{x}) \end{pmatrix}$$

where $\sin(2^i\pi\mathbf{x})$ is computed elementwise on $\mathbf{x}$. You might also consider concatenating the input $\mathbf{x}$ to the encoded points.

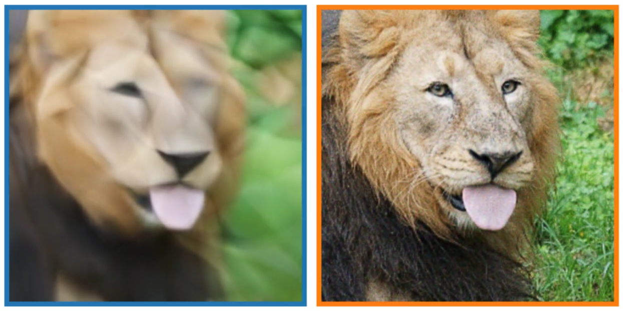


Figure 1: Image of a lion represented by two coordinate-based neural networks. On the left, the neural network has simply tried to memorize an image by mapping (x, y) coordinates to (r, g, b) colors. On the right, the (x, y) coordinate is instead represented with the positional encoding described above. Without the encoding, the network primarily learns low-frequency features. Even though the positional encoding provides no new information (each point is still entirely a function of its original 2D position), it is easier for the neural network to reason about each point's location in this expanded representation. Credit: Tancik et al.[4]

For example, with $n = 2$ and $L = 4$, this leads to a 16 dimensional representation of each 2D point. We will re-use this implementation for 3D points later, so make sure your implementation works regardless of the input dimension.

1.2 Fit a 2D image with MLP

The task involves fitting a 2D image using an MLP defined in the Model2d class. The model takes the positional encoding of 2D coordinates as input and predicts the RGB values at those coordinates. The model architecture comprises an input layer, a hidden layer, and an output layer, with ReLU activations between layers and a sigmoid activation at the output to ensure the RGB values remain in the range [0, 1].

The baseline implementation achieves a PSNR of around 26 after training for 10,000 iterations. To improve this, you are required to:

2 NeRF Model and Volume Rendering

In this section, you will be implementing a modified NeRF architecture and then building functions to use volume rendering to create an image of a new view based on the NeRF. All of the code for this will be in `src/part2.py`.

2.1 NeRF Architecture

Your first task will be filling in both the `__init__()` and `forward()` methods of the `NerfModel` class. NOTE: this is not the exact architecture from [2]. For simplicity, this assignment is forgoing two key characteristics of their approach — view dependency (meaning the model predictions will only depend on the world location of the point, not which angle you are viewing it from) and hierarchical sampling (more details in the subsection for sampling). It will consist of 8 fully connected layers and use ReLU activation functions. There will be a residual connection in the middle that will need to be handled carefully. Please follow more detailed instructions in the corresponding docstrings.

2.2 Ray Sampling

2.2.1 Get Rays

For a given camera view that we want to visualize, we can render an image by casting rays originating from the camera center and use the NeRF model to query color and density information at varying points along each ray. You will implement `get_rays()` function to take in image dimensions, the camera intrinsic matrix, and camera to transformation matrix and return the origins and directions of all of the *height* $\times$ *width* rays. The origins will actually all be the same (the camera center).

2.2.2 Ray Sampling

As mentioned above, in order to actually use the rays for rendering, we need to sample various points along the ray. The math of volume rendering is actually continuous and utilizes integrals, but for implementability we instead estimate these integrals using quadrature (think about the rectangle approximation methods for integrals you may have learned in calculus). Thus, we sample discrete points along the ray and composite their values together.

We will be using stratified sampling by partitioning the range from the near threshold t_n to the far threshold t_f into N bins and then uniformly sampling within the bins. This will be implemented in `sample_points_from_rays()`.

Mildenhall et al. [2] use a clever approach called hierarchical sampling which acknowledges that many of the points along the array will not actually help with the final image. They might be free space ($\rho = 0$) or occluded (transmittance $T(t) = 0$). Thus they first sample like above at a coarser level with a course network and use those outputs to choose good locations to query at a finer scale using a separate fine network. For simplicity, we will not be requiring you to implement hierarchical sampling but wanted to give this context.

2.3 Volume Rendering

This is the step where we use the results of the NeRF model from our query points to determine what to render at each pixel location. The math for this is described in great detail within the `proj6_local.ipynb` notebook, and there are implementation details in the docstrings of the corresponding functions, but at a high level we are trying to approximate the integrals

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

as the following summations

$$\hat{C}(\mathbf{R}) = \sum_{i=1}^{N} T_i(1 - \exp(-\sigma_i\delta_i))\mathbf{c}_i \qquad\qquad \hat{D}(\mathbf{R}) = \sum_{i=1}^{N} T_i(1 - \exp(-\sigma_i\delta_i))t_i$$

Here, transmittance is itself an integral $T(t) = \exp\left(-\int_{t_0}^{t} \sigma(s)\, ds\right)$ which is being approximated as $T_i = \Pi_{j=1}^{i-1}(1 - \alpha_j)$. The product $T_i(1 - \exp(-\sigma_i\delta_i))$ will be separately calculated as the compositing weight in `compute_compositing_weights()` before you put together all of these parts to sample rays and complete the volume rendering by calculating $\hat{C}(\mathbf{R})$ and $\hat{D}(\mathbf{R})$ in `render_image_nerf()`.

3 Train and use NeRF on Lego Dataset

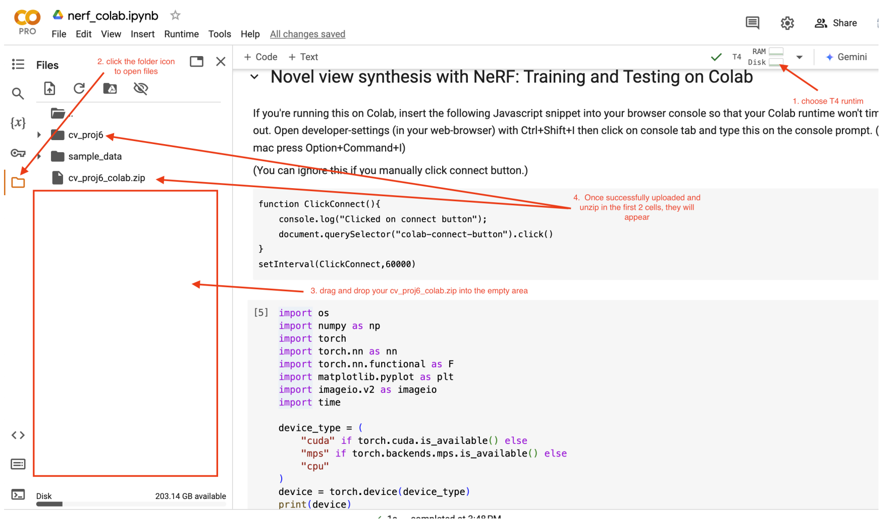
Now that you've worked with PyTorch and training/evaluating models throughout the past projects, you will apply this knowledge to fill in the gaps in the training code in *part3.py*. In this section, you will train a NeRF model on the Lego bulldozer dataset to synthesize novel views of the object. The goal is to implement missing parts of the training pipeline, fine-tune the model for accurate 3D scene reconstruction, and evaluate its performance by generating realistic images from unseen viewpoints.

This task integrates concepts of neural network training, volume rendering, and novel view synthesis, applying your knowledge of PyTorch and model evaluation techniques.

3.1 Training using Google Colab

- Execute “python zip_for_colab.py”, it will generates a zip file named “cv_project6_colab.zip”

- Upload “proj6_colab.ipynb” to your Google Drive, and open it using Google Colab.
- Change your Colab to use T4 GPU runtime.
- Once the runtime is ready, upload/drag and drop “cv_project6_colab.zip” to the file directory of current runtime. See Fig. 2.
- Then, you are good to start running the Colab cells.


nerf_colab.ipynb
PRO
File Edit View Insert Runtime Tools Help All changes saved
Share
+ Code + Text
T4
RAM
Disk
Gemini
Files
2. click the folder icon to open files
1. choose T4 runtim
cv_proj6
sample_data
cv_proj6_colab.zip
4. Once successfully uploaded and unzip in the first 2 cells, they will appear
3. drag and drop your cv_proj6_colab.zip into the empty area
Disk
203.14 GB available

Novel view synthesis with NeRF: Training and Testing on Colab

If you're running this on Colab, insert the following Javascript snippet into your browser console so that your Colab runtime won't tim
out. Open developer-settings (in your web-browser) with Ctrl+Shift+I then click on console tab and type this on the console prompt. (
mac press Option+Command+I)

(You can ignore this if you manually click connect button.)

```
function ClickConnect(){
    console.log("Clicked on connect button");
    document.querySelector("colab-connect-button").click()
}
setInterval(ClickConnect,60000)
```

```
[5] import os
    import numpy as np
    import torch
    import torch.nn as nn
    import torch.nn.functional as F
    import matplotlib.pyplot as plt
    import imageio.v2 as imageio
    import time

    device_type = (
        "cuda" if torch.cuda.is_available() else
        "mps" if torch.backends.mps.is_available() else
        "cpu"
    )
    device = torch.device(device_type)
    print(device)
```

Figure 2: Training using Colab.

4 Bells & whistles (extra *extra* credit)

Implementation of bells & whistles can increase your grade on this project by up to 5 points (potentially over 100). The max score is 105.

For all extra credit, be sure to include quantitative analysis showing the impact of the particular method you’ve implemented. Each item is “up to” some amount of points because trivial implementations may not be worthy of full extra credit. Some ideas:

- Up to 5 pts: The implementation of NeRF in this project leaves out a crucial ingredient — view dependency! With view dependency, the model would be queried with both the position of the point *as well as* the direction that you are viewing it from. As many real life surfaces don’t display Lambertian reflectance, this additional dependency allows the model to render specular reflections.
- Up to 5 pts: Hierarchical sampling can enable better efficiency. This concept is discussed in section 2.2.2 and more details are in the original paper.
- Up to 5 pts: MLPs struggle at capturing high frequency information or fine details in the signals they are learning [2, 3]. Mildenhall et al. [2] attempt to solve this by using positional encodings as explored in 1.1. However, Sitzmann et al. [3] argue that these issues can be at least partially attributed to the choice of ReLU as the activation function, and that a better approach would be to use periodic activation functions, namely sine in their proposed architecture SIREN. Chng et al. [1] build on this and propose Gaussian activation functions as another option in their proposed architecture GARF. Try using one of these alternative activation functions *without* positional encodings and see how it impacts the performance of your model as opposed to ReLU + positional encodings.

- Up to 5 pts: In this project we used an existing dataset of a Lego bulldozer. Alternatively, we can try to collect data on a real life scene ourselves and then train and use NeRF to render views of that. This would involve taking a video of a scene and then using a structure for motion solution like COLMAP to generate the necessary camera intrinsics info and camera to world coordinate transformation matrix. Try looking at the structure of the included Lego data and create something like that with your own data.

Rubric

- +80 pts: Code
- +20 pts: Report
- -5*n pts: Lose 5 points for every time you do not follow the instructions for the hand-in format. This includes failing to assign slides to problems you did not complete, including extra credit slides.

Submission format

This is very important as you will lose 5 points for every time you do not follow the instructions. You will submit two items to Gradescope:

1. `<your_gt_username>.zip` containing:
   (a) `src` - directory containing all your code for this assignment
   (b) `setup.cfg`: setup file for environment, no need to change this file
2. `<your_gt_username>_proj6.pdf` - your report

Do not install any additional packages inside the conda environment. The TAs will use the same environment as defined in the config files we provide you, so anything that's not in there by default will probably cause your code to break during grading. **Do not** use absolute paths in your code or your code will break. Use relative paths like the starter code already does. Failure to follow any of these instructions will lead to point deductions.

Create the zip file using:

```
python zip_submission.py --gt_username <your_gt_username>
```

(It will zip up the appropriate directories/files for you!) and hand it in with your report PDF through Gradescope (please remember to mark which parts of your report correspond to each part of the rubric).

Credits

This assignment is based on a project from Noah Snavely's Computer Vision course at Cornell, with modifications and additions made by James Hays, Akshath Anna, Jim James, Akshay Krishnan, Meena Nagarajan, Mengyu Yang, and Eric Zhang.

References

[1] Shin-Fang Chng, Sameera Ramasinghe, Jamie Sherrah, and Simon Lucey. Garf: Gaussian activated radiance fields for high fidelity reconstruction and pose estimation, 2022. URL https://arxiv.org/abs/2204.05735.

[2] Ben Mildenhall, Pratul P. Srinivasan, Matthew Tancik, Jonathan T. Barron, Ravi Ramamoorthi, and Ren Ng. Nerf: Representing scenes as neural radiance fields for view synthesis, 2020. URL https://arxiv.org/abs/2003.08934.

[3] Vincent Sitzmann, Julien N. P. Martel, Alexander W. Bergman, David B. Lindell, and Gordon Wetzstein. Implicit neural representations with periodic activation functions, 2020. URL https://arxiv.org/abs/2006.09661.

[4] Matthew Tancik, Pratul P. Srinivasan, Ben Mildenhall, Sara Fridovich-Keil, Nithin Raghavan, Utkarsh Singhal, Ravi Ramamoorthi, Jonathan T. Barron, and Ren Ng. Fourier features let networks learn high frequency functions in low dimensional domains. *NeurIPS*, 2020.