University of Michigan
EECS 442: Foundations of Computer Vision
Fall 2023. Instructor: Andrew Owens.

Problem Set 9: Neural Radiance Fields

Posted: Monday, Nov. 27, 2023 **Due: Wednesday**, Dec. 6, 2023

Please convert your Colab notebook to a PDF file and submit the PDF file to Gradescope, *making sure to label your answers*. We have included the PDF conversion script at the end of the notebook. Nothing needs to be submitted to Canvas. **Credit:** The code of this problem set is largely taken from the original implementation of the Fourier Feature Networks paper and the NeRF from Nothing tutorial.

The starter code can be found at:
https://drive.google.com/file/d/1VnwGKCKjX-Jo_8NFrl3asPrMNWBo1ZGG/view?usp=sharing

We recommend editing and running your code in Google Colab, although you are welcome to use your local machine instead.

Problem 9.1 *Fitting an MLP to a single image*

As a starting point, we will train a multilayer perceptron (MLP) to reproduce a given input image. The MLP will take, as input, the (x, y) coordinate of a pixel as input, and will directly predict its corresponding (r, g, b) color.

We will study the effect of using a positional encoding to represent the input. Given a vector $x \in \mathbb{R}^2$, the full positional encoding is: $\gamma(x) = (\sin(2\pi x), \cos(2\pi x), \cdots, \sin(2^n \pi x), \cos(2^n \pi x))$, where cos and sin are applied pointwise to their input (i.e., applied to each coordinate of the vector independently). In our implementation, we will consider different values of n, thereby controlling the number of frequencies that are used in the representation. We will control this using a matrix B, such that

$$\gamma(x) = \left(\cos\left(2\pi x B^\top\right), \ \sin\left(2\pi x B^\top\right)\right). \tag{1}$$

By choosing different values for B, we can thereby control the number of frequencies that are used by the model.

(a) **(2 points)** Implement the function `input_mapping(self, x, B)` in 9.1.1 that applies a Fourier feature mapping, specified by B, to the input x.

(b) **(1 points)** Define the base Fourier feature matrix B in 9.1.4 that encodes input vector x into $\gamma(x) = (\sin(2\pi x), \cos(2\pi x))$. Your results should be similar to those in Fig. 1.

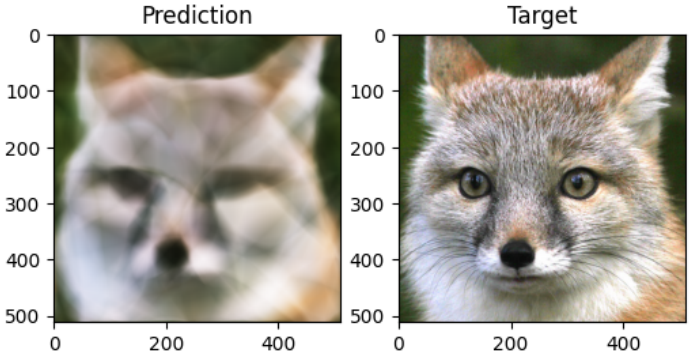

Figure 1: Example prediction with base Fourier features.

(c) **(1 points)** Define the full Fourier features matrix B in 9.1.5 that encodes input x into $\gamma(x) = (\sin(2\pi x), \cos(2\pi x), \cdots, \sin(2^n \pi x), \cos(2^n \pi x))$. Your results should be similar to those in Fig. 2.

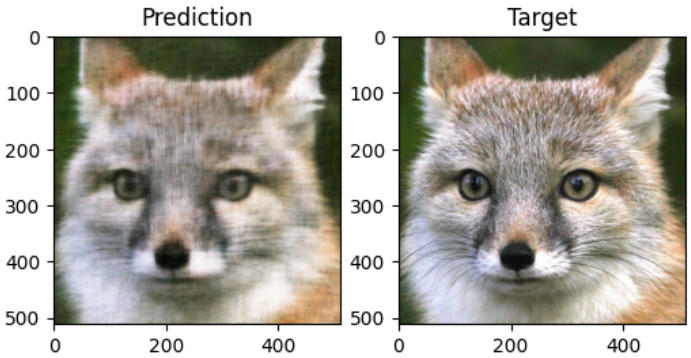

Figure 2: Example prediction with full Fourier features.

Problem 9.2 *Neural radiance fields*

We will fit a neural radiance field (NeRF) to a collection of photos (with their camera pose), and use it to render a scene from different (previously unseen) viewpoints. To estimate the color of a pixel, we will estimate the 3D ray that exist the pixel. Then, we will walk in the direction of the ray and query the network at each point. Finally, we will use volume rendering to obtain the pixel's RGB color, thereby accounting for occlusion.

Recall that the NeRF is similar to the MLP that you implemented in 9.1. It is an MLP F_Θ such that

$$F_\Theta(x, y, z, \theta, \phi) = (R, G, B, \sigma), \tag{2}$$

where (x, y, z) is a 3D point in the scene, and (θ, ϕ) is a viewing direction. It returns a color (R, G, B) and a (non-negative) density σ that indicates whether this point in space is occupied.

(a) **(1 points)** Implement the function `positional_encoder(x, L_embed = 6)` in 9.2.2 that encodes the input x as $\gamma(x) = (x, \sin(2^0 x), \cos(2^0 x), \cdots, \sin(2^{L\_embed-1} x), \cos(2^{L\_embed-1} x))$. This will be similar to your implementation of 9.1.

(b) **(3 points)** Implement the code that samples 3D points along a ray in 9.2.5. This will be used to march along the ray and query F_Θ.

(c) **(2 points)** After having walked along the ray and queried F_Θ at each point, we will estimate the pixel's color, represented as `rgb_map` (in 9.2.5). We will also compute, `depth_map`, which indicates the depth of the nearest surface at this pixel.

We can now render the NeRF from different viewpoints. If everything implemented correctly, your rendered results from camera pose #40 will be similar to Fig. 3.

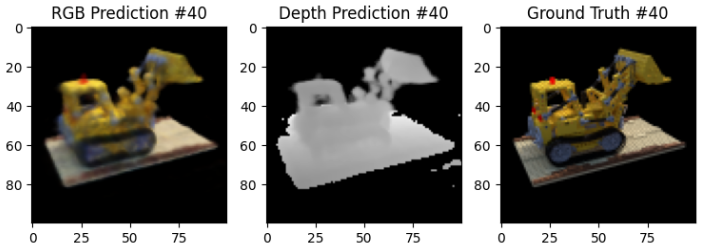

Figure 3: Example prediction from the NeRF.