NeRF: Neural Radiance Fields

Computer Vision (RBE549) Project 2

Hrishikesh Pawar
MS Robotics Engineering
Worcester Polytechnic Institute
Email: hpawar@wpi.edu

Tejas Rane
MS Robotics Engineering
Worcester Polytechnic Institute
Email: turane@wpi.edu

***Abstract*—The report outlines the end-to-end implementation of the NeRF paper.**

I. Phase 2: Neural Radiance Field (NeRF)

In this phase, the primary objective is to implement the NeRF model [1], enabling the reconstruction of a volumetric scene representation through the combination of a corpus of images obtained from different vantage points. Our execution for NeRF is outlined as follows:

1) Data Pre-processing.
2) Network Architecture - Multi-Layer Perceptron (MLP).
3) Volume Rendering
4) Training Implementation Details.
5) Results and Discussion.

Subsequent sections detail the methodologies adopted within each segment.

A. Data Pre-processing

Datasets Utilized:

1) Lego
2) Ship

The chosen datasets comprise an array of images alongside their corresponding camera poses, encapsulated through transformation matrices incorporating the spatial translation and rotation from the camera coordinate framework to the world coordinate system.

The data pre-processing stage employs classical volume rendering techniques, wherein each pixel in the images is interpreted as a ray within the three-dimensional space. The transformation from pixel coordinates (u, v) to normalized coordinates $(X, Y, 1)$ with respect to the camera's center is initialized. The ray equation is defined as follows:

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

Here, o represents the origin of the ray, corresponding to the pixel location in the three-dimensional space, d denotes the ray direction as a unit vector from the camera's center to the image pixel, and t is a parameter of the equation, sampled at intervals. Leveraging the rotation matrix derived from the transformation matrix representing the camera-to-world rotation and translation, the ray direction is transformed into the world coordinate system and subsequently normalized to obtain a unit vector. The transformation of the image rays is depicted in Fig. 1 [2]

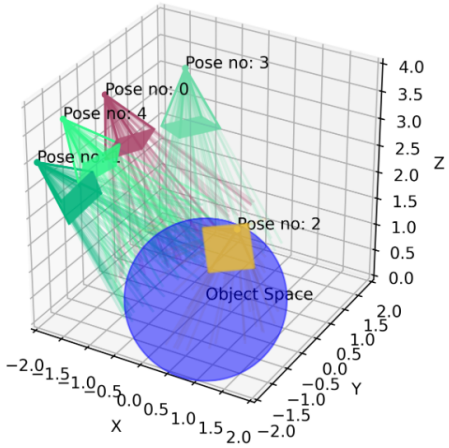

Fig. 1: Rays transformation from camera to world coordindates

To facilitate model input post ray generation, we implemented the following data architecture:

1) **Ray Positions**: Expressed as a tensor of dimensions $N \times (H \times W) \times 3$, where N signifies the number of images, while H and W denote the image height and width, respectively.
2) **Ray Directions**: Analogously, a tensor of size $N \times (H \times W) \times 3$ encapsulates the direction vectors.
3) **Target Pixel Values**: The corresponding color values are encapsulated within a tensor of dimensions $N \times (H \times W) \times 3$.

These tensors were aggregated to form the final model input, a concatenated tensor of shape $N \times (H \times W) \times 6$, merging ray positions (first three dimensions) and directions (next three dimensions), which were subsequently segmented into batches for the training process.

B. Network Architecture - Multi-Layer Perceptron (MLP)

We employed a Multi-Layer Perceptron (MLP) with a series of fully connected layers, designed to process continuous 5D coordinates, comprising spatial location (x, y, z) and viewing

[1] NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis)

[2] Deep Dive into NeRF (Neural Radiance Fields)

direction (θ, ϕ) to predict volume density and view-dependent emitted radiance. Refering to the model architecture in Fig. 11 key aspects are as follows:

- The architecture utilizes embeddings for spatial position and viewing direction, denoted as $\mathbf{L}_{pos}$ and $\mathbf{L}_{dir}$, set to 10 and 4 respectively. These values enable the MLP to discern high-frequency features within the scene.
- Initially, the input layer enhances the input vector's dimensionality using positional encoding. Following this enhancement, a sequence of fully connected layers processes the spatial information. Post these initial layers, the viewing direction is concatenated to the feature vector for subsequent processing and radiance prediction.
- The MLP is characterized by hidden layers with dimensions of 256 and 128 for initial and concatenated layers, respectively.
- Ultimately, the network bifurcates its output to deliver volume density σ and RGB color. It employs ReLU and sigmoid activations for each output type to maintain the physical validity of these predictions.

1) Positional Encoding: Positional encoding is pivotal in allowing the NeRF model to capture high-frequency details effectively:

$$\begin{aligned}\gamma(\mathbf{x}) = [&\sin(2^0\pi\mathbf{x}), \cos(2^0\pi\mathbf{x}),\\ &\ldots, \sin(2^{L-1}\pi\mathbf{x}), \cos(2^{L-1}\pi\mathbf{x})]\end{aligned} \tag{2}$$

where L reflects the frequency bands, chosen separately for spatial ($\mathbf{L}_{pos}$) and directional ($\mathbf{L}_{dir}$) embeddings. This encoding augments the MLP's ability to detect and utilize minute variations within the input, aiding in precise scene reconstruction.

C. Volume Rendering

The crux of visualizing the NeRF model's output is the volume rendering algorithm, which transforms the neural representation into a 2D image. The process is outlined as follows:

1) **Ray Sampling:** Discretize each ray from camera origin o through the scene within the bounds $[t_n, t_f]$ into N segments and perturb these samples.
2) **Density and Color Estimation:** For each sample, the MLP predicts the RGB color and volume density, offering a discrete approximation of the scene's radiance field.
3) **Transmittance Computation:** Evaluate the accumulated transmittance $T(t)$ along the ray to determine how much light reaches each point.
4) **Radiance Accumulation:** Aggregate the contributions of all samples along the ray to compute the final pixel color, considering both direct radiance and accumulated transmittance.

Formally, the expected color $C(r)$ captured by the camera for a given ray r is computed using the integral:

$$C(r) = \int_{t_n}^{t_f} T(t)\sigma(r(t))c(r(t), d)dt \tag{3}$$

where $T(t) = \exp\left(-\int_{t_n}^{t} \sigma(r(s))ds\right)$ represents the transmittance. We employ stratified sampling for numerical estimation:

$$\hat{C}(r) = \sum_{i=1}^{N} T_i(1 - \exp(-\sigma_i\delta_i))c_i \tag{4}$$

where $\delta_i = t_{i+1} - t_i$ denotes the segment length, and T_i and c_i are the transmittance and color at each sample. This integration strategy ensures a continuous and detailed rendering of the scene.

By embedding these mechanisms into the NeRF model, we synthesized images from novel viewpoints.

D. Training Implementation Details

The implemented training regimen refines the neural network parameters to minimize the loss between the rendered images and the corresponding ground truth pixel values.

1) Loss Function Computation: The loss function central to our model's training is the mean squared error (MSE) between the predicted RGB values and the ground truth pixel values for a set of rays:

$$\mathcal{L} = \frac{1}{N}\sum_{i=1}^{N} \left\| C(r_i) - \hat{C}(r_i) \right\|^2, \tag{5}$$

where $C(r_i)$ and $\hat{C}(r_i)$ represent the ground truth and predicted color for the i-th ray, respectively, and N is the number of rays in the batch.

2) Training Hyperparameters: Training hyperparameters are outlined as follows:

- **Epochs:** 20
- **Batch Size:** 1024
- **Near Bound t_n:** 2
- **Far Bound t_f:** 6
- **Samples along ray:** 192
- **Learning Rate (lr):** 5×10^{-4}
- **Optimizer:** Adam
- **Learning Rate Scheduler:** Employed with a gamma value of 0.5 and milestones set at epochs 2, 4, and 8 to adjust the learning rate adaptively.

The training progression is depicted in Fig. 10

E. Results and Discussion

The training process was executed on two distinct datasets: Lego and Ship with image dimensions to be 400×400. Utilizing an A30 computational resource, the model underwent training spanning over 315k iterations, which amounted to approximately 22 hours of training time. This process included the implementation of positional encoding to enhance the model's ability to capture high-frequency details. Fig 2 and 3 show the training PSNR and training loss plotted against the number of iterations (x1000). We also attempted training the model with the complete 800×800 resolution of the image, but that took approximately 27 hours for just 128k iterations. Table I encapsulates the Structural Similarity Index Measure

(SSIM), and Peak Signal-to-Noise Ratio (PSNR) across test dataset comprising of 200 images.

Dataset	SSIM	PSNR
Lego	0.87	29.885
Ship	0.83	28.145

TABLE I: Average SSIM, and PSNR values

We also tried training the NeRF model without positional encoding. We observed that the training does not converge, and might need further parameter tuning to generate the desired results. Fig 4 and 5 show the training PSNR and training loss plotted against the number of iterations (x1000) for the training trial without positional encoding.

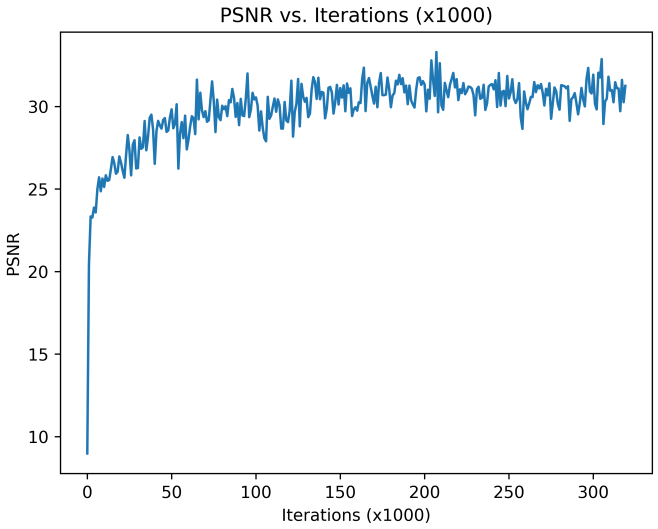

Fig. 2: Training PSNR vs Iterations (x1000), with positional encoding, Lego dataset

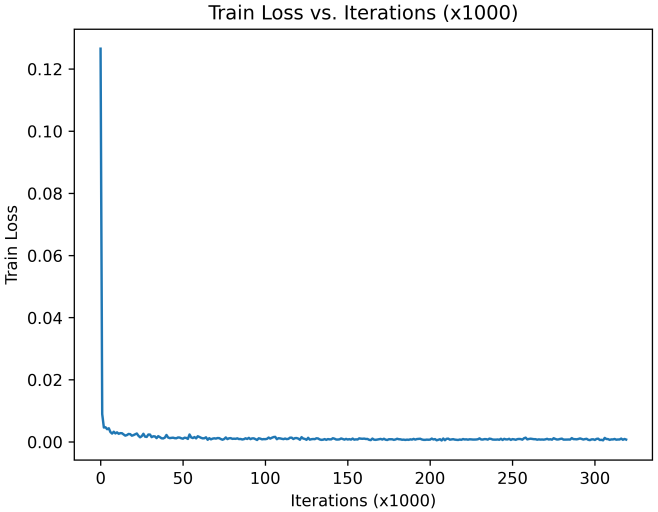

Fig. 3: Training Loss vs Iterations (x1000), with positional encoding, Lego dataset

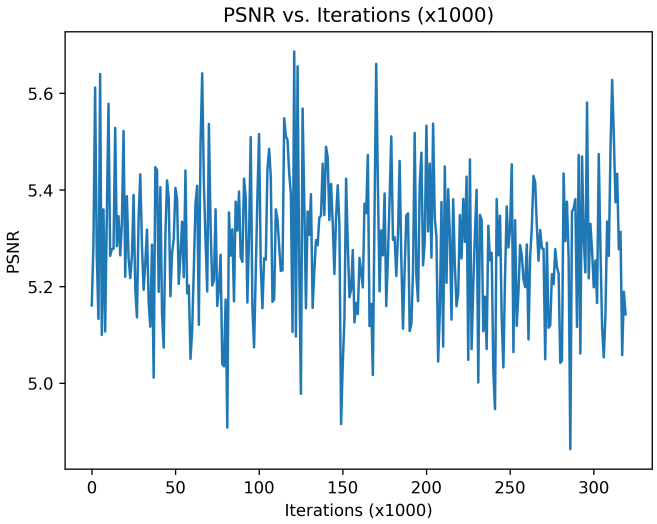

Fig. 4: Training PSNR vs Iterations (x1000), without positional encoding, Lego dataset

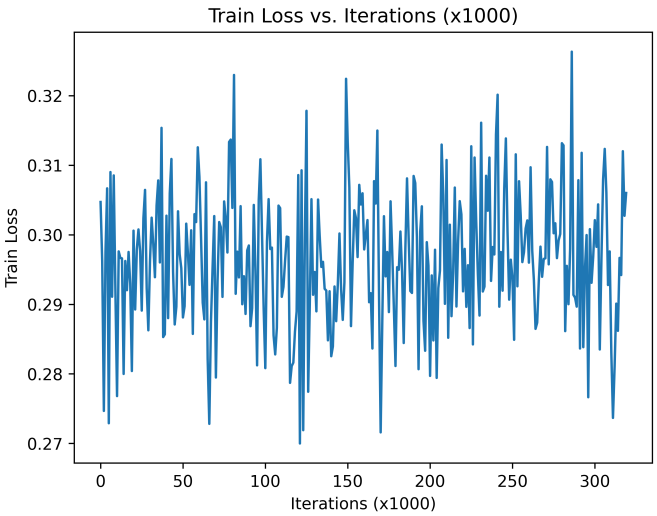

Fig. 5: Training Loss vs Iterations (x1000), without positional encoding, Lego dataset

Fig. 6 depicts the novel views for the lego dataset and Fig. 7 depicts the novel views for the ship dataset. Fig. 8 and Fig. 9 depicts the comparison of the test dataset with our rendered novel views.

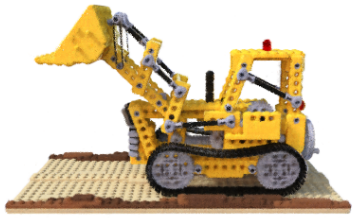

Fig. 6: Lego Dataset Novel Views (generated from out model)

Fig. 7: Ship Dataset Novel Views (generated from out model)

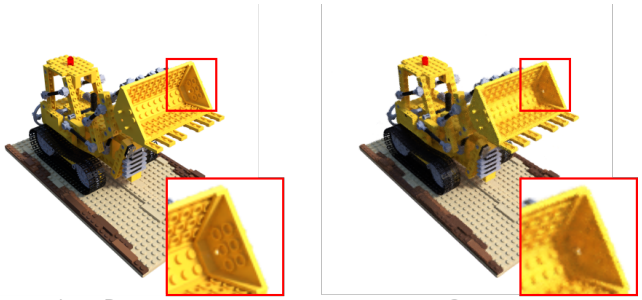

Fig. 8: Lego Dataset Comparison

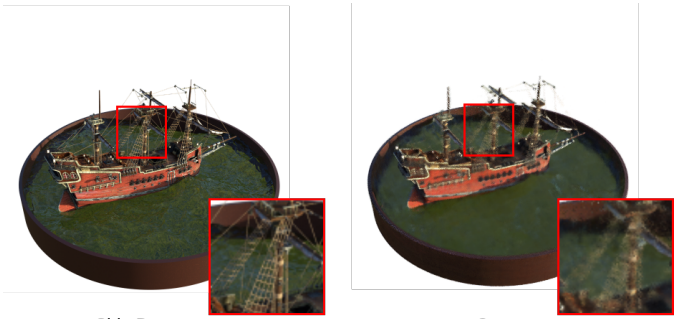

Fig. 9: Ship Dataset Comparison

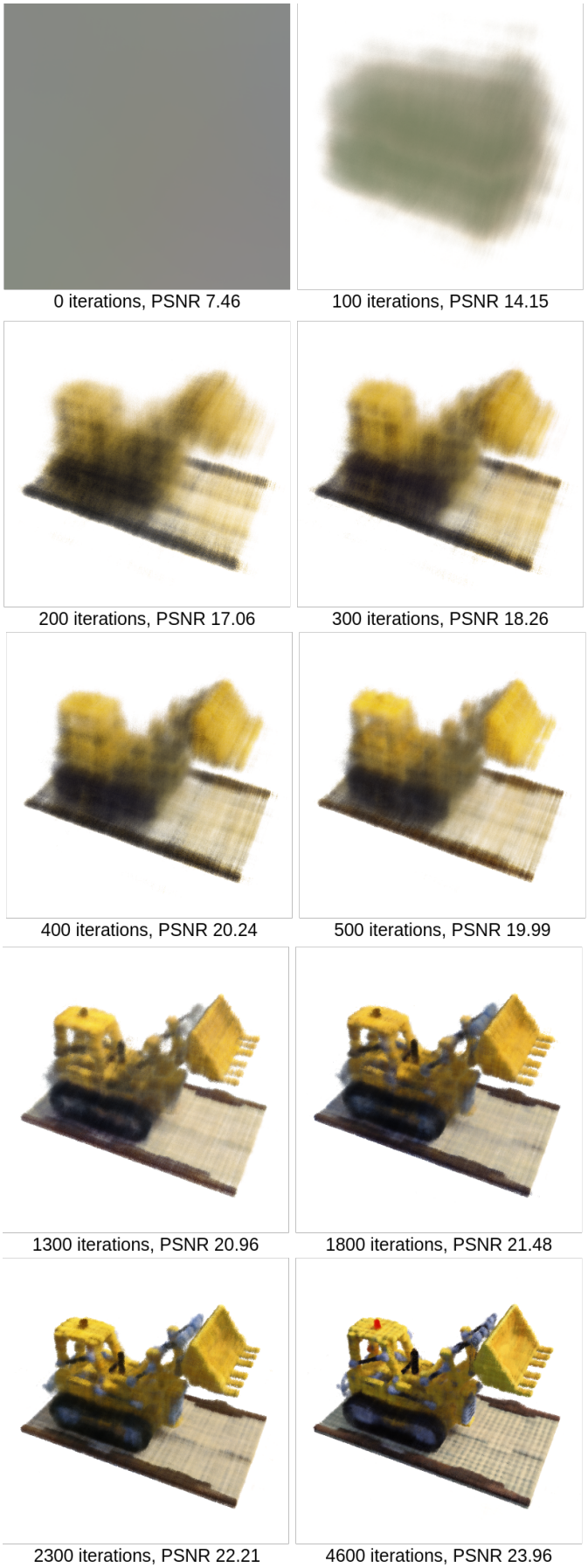

Fig. 10: Training Progression

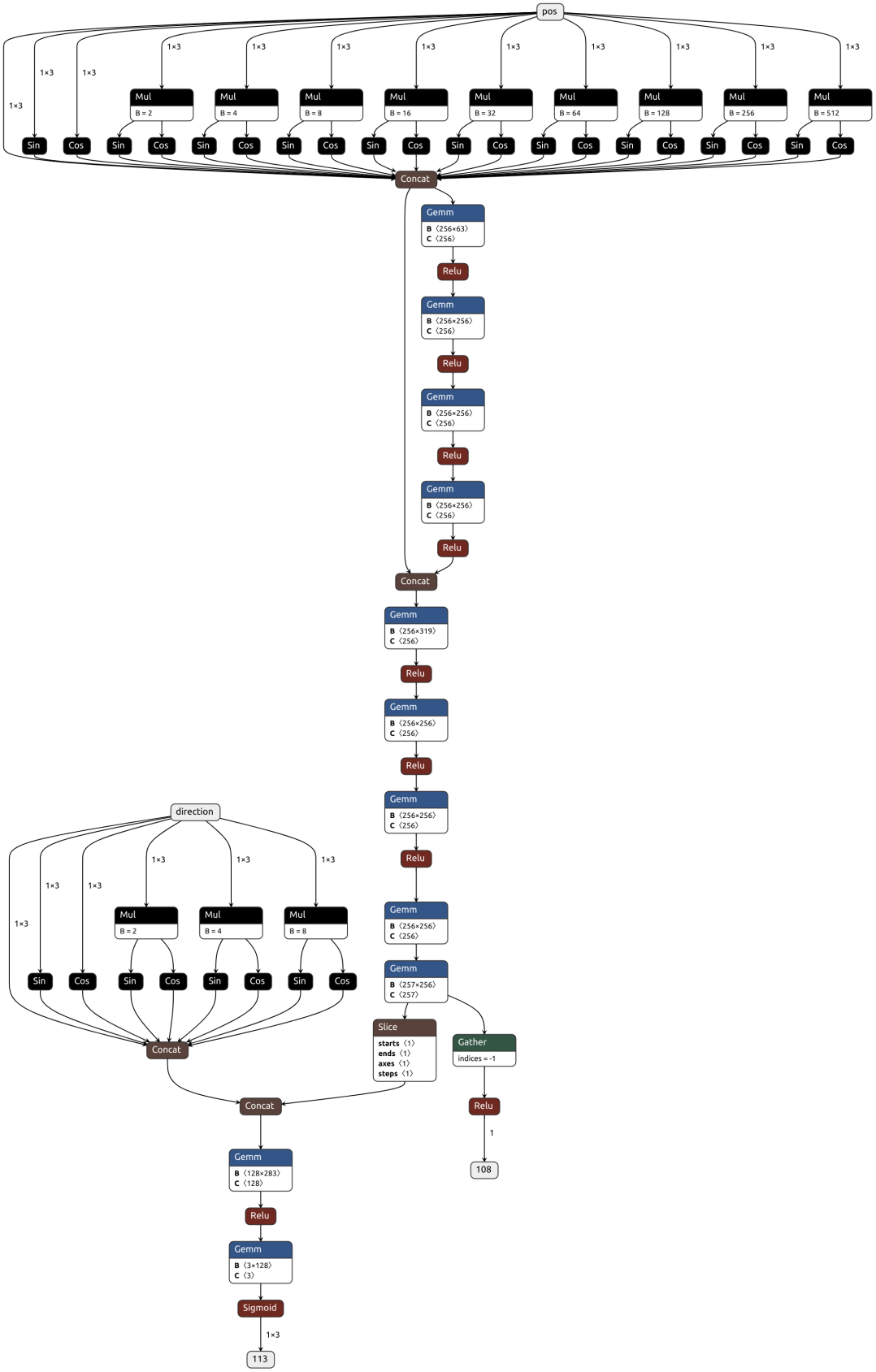

Fig. 11: Model Architecture