Efficient 3D Reconstruction

Project Final Report (CS8803 - Efficient Machine Learning)

Vijayraj Shanmugaraj

Neural Radiance Fields (NeRFs) were a groundbreaking step in the field of 3D rendering, learning to model a scene as a complex function of viewpoint coordinates, light and material properties, and synthesize highly realistic images from sparse sets of 2D photographs. However, rendering a single scene takes the NeRF ~30 seconds, which would be too long for realtime applications such as robotic perception and AR/VR.

Previous work

Some select papers on making NeRFs efficient are as follows:

- **Attal et al. (2022):** "Ray Space Embedding Networks" introduces a method to learn a subdivided representation of a given scene for more efficient handling of sparse inputs and complex view-dependent effects, reducing the computational and memory requirements.
- **Wu et al. (2023):** This paper proposes "NeRDFs", models the radiance distribution along rays parameterized using a set of trigonometric functions, and synthesizes images with volumetric rendering. NeRDFs also requires only one single network forward pass per pixel for view synthesis.
- **Xie et al. (2023):** "HollowNeRF", introducing a novel compression solution that learns to prioritize visible surface features and prune invisible internal features without needing prior knowledge of surface geometries an an optimization framework for pruning unnecessary features during NeRF training.

Limitations of existing work

Existing works seem to focus mainly on modifying the NeRF architectures, rather than implementing simpler methods and observing their effect on the NeRF rendering, tending to be a bit complex. Additionally, none of the methods take into account the nature of the activations from the different layers of the NeRF, and attempt at pruning accordingly.

Proposed solution

Experiment 1: Quantization

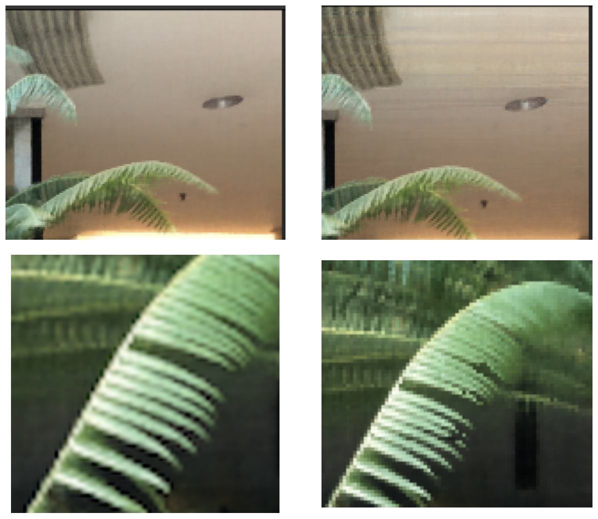
As a part of this project, I would be trying out various standard techniques such as pruning, quantization, and other training routines combined with previously implemented solutions to shrink the model size of NeRFs to enable faster inference time.

I implemented basic quantization techniques and activation-based pruning methods to reduce the model size. First I tried quantizing the NeRF using classical methods. We use PyTorch's Post-Training Quantization framework with a CPU backend. The weights are quantized to qint8 in a symmetric manner i.e. (-127, 128). The results from the quantization experiment are as follows:

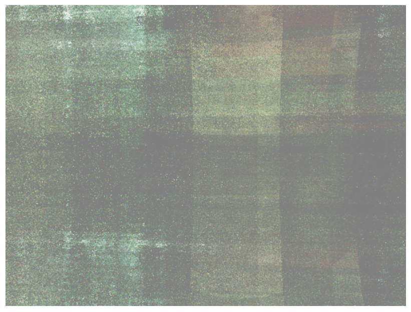
	PSNR	Rendering time	Model Size
Original	26.503	10.2 min/scene	2.27MB
8-bit Quantized (Both fine and coarse)	21.173	6.83 min/scene **(-33%)**	1.24MB **(-45%)**
8-bit Quantized (Fine only)	22.435	8.12 min/scene **(-20.4%)**	1.76MB **(-22%)**
8-bit Quantized (Coarse only)	22.167	8.23 min/scene	1.76MB **(-22%)**

Analysis of results

We see that simple quantization of the model has brought down the size by about 45%, with comparatively minimal effect on accuracy. We however see that the depth perception of the model becomes poorer with quantization, mainly affecting the elements at a higher depth, becoming distorted (ex. The floor and the ceiling). A sample scene, along with features have been shown below.



(a)

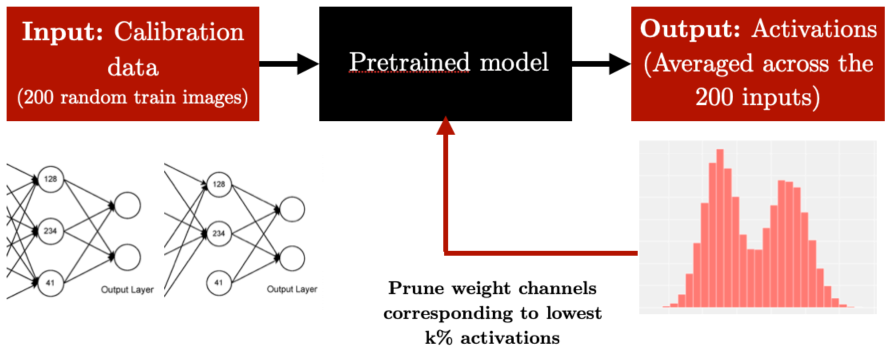


(b)

Rendering examples: Above, we can see the accuracy with which the 8-bit model (right images in (a)) render with respect to the original model.

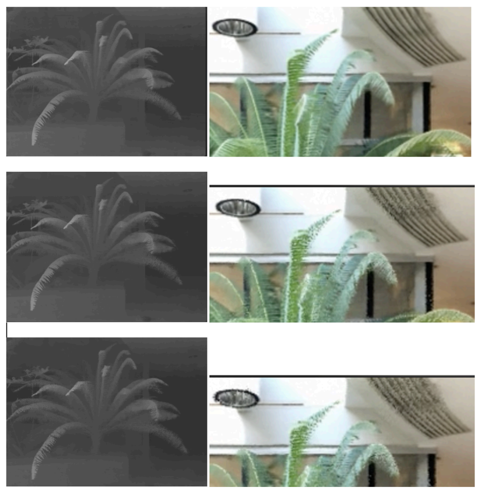
Occasionally, the rendering of the 8-bit model seems to fail completely, leading to an incoherent output (b). Although the model usually renders with decent accuracy, such failures for certain poses would make the model useless for applications like 3D rendering. This would call for a more careful pruning technique to ensure proper rendering for all inputs. **Pruning the fine and coarse networks seem to give similar results as the normal quantization.**

Experiment 2: Activation based post-training pruning
Previous works for NeRFs tinker with the MLP architectures, but don't take the activations into account. I build on this to create a new framework for pruning NeRFs. The proposed pruning methodology is as follows:


Input: Calibration data (200 random train images)
Pretrained model
Output: Activations (Averaged across the 200 inputs)
128
234
41
Output Layer
128
234
41
Output Layer
Prune weight channels corresponding to lowest k% activations

Essentially, we prune the NeRF on the basis of the importance of the activations generated by weights on the calibration data.

Pruning %	Model size	PSNR	Loss
0	2.27MB	26.503	0.086
0.15	1.99MB	23.934	0.071
0.3	1.72MB	23.932	0.073
0.5	1.34MB	22..61	0.853
0.85	0.75MB	22.5	0.102



Analysis of results

Left: Tabulated results for different pruning percentages, **Right:** image results (from top to bottom): 0.15, 0.5, 0.85.

We can see that pruning does not affect PSNR too much, however the loss increases progressively as the model is pruned. The model slowly loses clarity with increase in pruning percentage, however, we see that the model loses coherence as the pruning percentage increases, distorting both foreground and background elements. The model hits a sweet spot at 0.3. The consistent PSNR may be due to the fact that **PSNR scores do not always correlate with perceived quality**. One of the most common areas where it has been observed that PSNR does not always represent the perceived quality of an image is blurriness. The blurry nature of the outputs may be a contributing factor to the consistent PSNR.

Future work

Quantization: Quantizing layers based on their activations can be a viable option (just like the activation-based pruning), and variable quantization of the fine and coarse networks to see whether quality can be retained.

Pruning: Entropy-based methods to prune weights across the network instead of pruning all the weights associated with an activation would be a valuable extension to the work being done here.

Expected outcomes

Reconstruction time: I brought down the reconstruction time of the NeRF model on a scene down by at least 50%. Although the ideal goal would be to have near-realtime rendering of a given scene, I managed to get commensurate results with a 50% drop in model size and rendering time.