Improved Camera Pose Estimation for NeRFs with CNNs and Curriculum Learning

Marcel Torne Villasevil
Massachusetts Institute of Technology
marcelto@mit.edu

Florian Juengermann
Massachusetts Institute of Technology
florianj@mit.edu

Abstract

Neural Radiance Fields (NeRF) solve the novel view synthesis problem by using a differentiable renderer to train a neural network to learn a 3d scene representation. The initially proposed method requires knowledge of the exact camera positions and orientations for all input images. Recent work [11] tried to loosen that constraint by jointly optimizing the camera poses and 3d representation. However, this approach only works for forward-facing scenes where all pictures have significant overlap. This work tries to extend this method to 360-degree reconstructions by iteratively learning more viewing angles with curriculum learning. For that, we assume to have a broad ordering of input pictures according to the camera's position.

We also introduce a new method of learning the camera parameters. Instead of independently optimizing the parameters for each training image as in [11], we use a convolutional neural network (CNN) to predict the camera position from the training image. Finally, we evaluate the practicality of this approach by evaluating the methods on a new dataset captured with a smartphone. [1]

1. Introduction

Neural Radiance Fields (NeRFs) are deep neural networks that learn 3D scene representations enabling novel view synthesis. For a given position (x, y, z) in space, the model outputs the tuple (r, g, b, a) denoting the color and opacity at that location. To render a $2d$ image, for every pixel, we cast a ray from the camera position into the scene and sample the model along that ray to receive the color value. If we know the exact camera pose of the input images, we can train the network to output the input image from that position. The key to why this works is the differentiable rendering process that allows for training the model with backpropagation. After the training, it is possible to render views from any camera position. This technique has applications in various fields, we give an example of the sim2real problem in robotics.

Applications in sim2real One of the biggest problems in learning controllers for robotics is the problem of sim2real transfer. Nowadays, most controllers in robotics are learned using Reinforcement Learning and these are trained in simulation since they need abundant data. However, simulators are not accurate representations of how the real world behaves. For this reason, when an RL model is trained in a simulator, if we later deploy it in real life most of the time it does not behave as well as in the simulator since its environment changed. There have been many proposed techniques to mitigate this problem but this is still an active area of research. From the Computer Vision standpoint, one of the problems in the sim2real transfer is that the rendering in the current simulators is far from an accurate depiction of reality. However, we can envision simulators that use NeRFs to represent a scene from real life that has been recorded previously then we would have very accurate representations of real environments and this could be a big step toward solving the sim2real transfer problem in the vision part of the controller. For example, researchers at Waymo trained a NeRF that represents the city of San Francisco [10]. We can see that in this case, we could train an autonomous driving controller on a simulator that uses this NeRF. Other interesting applications include for example obtaining models of objects with just a set of pictures instead of having to build the CAD models which require much more time and human labor.

Limitations The initially proposed method already achieves great visual results. Its main limitations concern long training times, restrictions to non-dynamic scenes, and the requirement to know the exact camera poses. Our work addresses the last issue. For many real-world applications, the exact camera positions are not known unless specialized equipment is used. Our goal is to reconstruct the 3d rep-

[1] The code for this paper can be found at: https://github.com/florianjuengermann/CurriculumNeRF.git

resentation from nothing more than a sequence of pictures taken from a mobile phone that is roughly ordered according to some position proximity metric.

2. Related Work

Since [7] introduced NeRFs, they have been extended by a suite of works. We highlight a small selection relevant to our approach.

Performance improvements [5] and [13] use special data structures to speed up the rendering process. Specifically, by dividing the 3d space up into cubes of adaptive sizes octree data structure allows us to quickly find non-transparent parts. Other work [10] made it possible to represent large scenes by dividing the scene into independent blocks.

Unknown camera parameters Traditionally, SLAM methods [2, 3, 8] are used to map an environment without knowing the camera position and orientation. [12] estimates the camera pose with a pre-trained NeRF. In line with the goal of this paper, [11] jointly trains a NeRF together with the pose estimation. However, their approach is limited to forward-facing images which makes the pose estimation easier compared to full 360-degree capture. In this work, we try to improve the convergence speed for complex scenes and more diverse viewing angles using curriculum learning.

3. Neural Radiance Fields Architecture and Optimizations

In this section, we briefly recap how neural radiance works, which architecture they use, and baseline optimizations that are needed. The NeRF model is a simple feed-forward neural network that maps space coordinates to color and opacity values. For our experiments, we use eight fully connected layers with 200 nodes each with ReLU activations. In the following paragraph, we discuss common extensions to this architecture.

Viewing Angle Dependence Instead of only depending on the space coordinates (x, y, z), [7] proposes to also input the viewing angle θ, ϕ into the network. This way, we can capture effects such as reflections and glares. However, the viewing angle should not impact the opacity at a coordinate. So [7] only input (x, y, z) in the beginning, predict the opacity, and only then input the viewing angle to predict the color values. In our work, we did not model this behavior as we focused on generating a course 3d representation in difficult conditions rather than the most brilliant result.

Positional Encoding While it is known that neural networks are universal function approximators, in practice, they oftentimes learn low-frequency signals [9]. To represent 3d objects in high fidelity, these high frequencies are crucial. As proposed in [7], we explicitly encode our input coordinates with Fourier features:

$$\text{encPos}(\xi) = [\sin(2^0\xi), \cos(2^0\xi), \dots, \tag{1}$$

$$\sin(2^{L-1}\xi), \cos(2^{L-1}\xi)] \tag{2}$$

Additionally, we found it helpful to input the position again in the middle of the network. So we concatenate the layer-4 features with the position as input into the next layer.

Volumetric Rendering For each pixel in the output image, we want to determine the color. For a fixed camera position and rotation, one pixel is identified by a ray $\mathbf{r}$ from the camera into the scene. Along this way, we query the NeRF to obtain color values $\mathbf{c}(\mathbf{r}(t))$ and opacity values $\alpha(\mathbf{r}(t))$. The opacity can be interpreted as the probability of a ray being absorbed in that location. Formally, the pixel color is defined by the volumetric rendering formula:

$$I(\mathbf{r}) = \int_{t_n}^{t_f} T(t)\alpha(\mathbf{r}(t))\mathbf{c}(\mathbf{r}(t))\mathrm{d}t \tag{3}$$

where

$$T(t) = \exp\left(-\int_{t_n}^{t_f} \alpha(\mathbf{r}(s))\mathrm{d}s\right) \tag{4}$$

describes the probability that a ray travels from t_n to t without being absorbed.

We approximate this integral by sampling points in the interval $[t_n, t_f]$ and using a quadrature method as shown in [7]:

$$\hat{I}(\mathbf{r}) = \sum_{i=1}^{N} T_i(1 - \exp(-\alpha_i \cdot (t_{i+1} - t_i))\mathbf{c_i} \tag{5}$$

with

$$T_i = \exp\left(-\sum_{j=1}^{i-1} \alpha_j \cdot (t_{j+1} - t_j)\right) \tag{6}$$

and discrete samples $\{\alpha_i\}, \{\mathbf{c}_i\}$.

4. Camera Parameter Estimation

Most work regarding NeRFs considers the case where the exact camera position and orientation are known for each input image. As shown in [11], in certain settings, it is possible to train the NeRF without this information.

Camera Parameterization A camera pose in 3d is an object with 6 degrees of freedom. We use a parameterization with an rotation vector $\mathbf{r} \in \mathbb{R}^3$ and a location vector $\mathbf{t} \in \mathbb{R}^3$. We can think of these vectors as describing the transformation from a camera at the origin with a fixed orientation. First, we rotate the camera around an axis $\hat{\mathbf{r}}$ by the angle determined by $\|\mathbf{r}\|$. Then, we translate the camera to position $\mathbf{t}$. These six parameters are going to be the trainable camera parameters θ. Previous work also considered learning the focal length f [11], however, for our purposes, we assume the focal length to be known in advance as that is the case in most real-world applications.

When working with the camera parameters, it turns out to be useful to define a transformation matrix that captures both $\mathbf{r}$ and $\mathbf{t}$. Following previous work, we use a parameterization in homogeneous coordinates yielding a 4×4 transformation matrix.

4.1. Joint Optimization

Wang et al. [11] showed that it is possible to learn the camera parameters θ_i for each training image i simultaneously to train the NeRF for forward-facing scenes. Their model consists of a simple look-up table independently storing camera parameters θ_i for each training image index i. When the network is trained on image i, the gradient of θ_i with respect to the loss is calculated through the differentiable rendering process with backpropagation. Then the parameters are adjusted accordingly. This method is dependent on not having too large variations in the camera angle. For large changes in the angles of the scene, this method will most likely fail. The reason is that the parameters are initialized at the identity matrix, needing many large steps in backpropagation to converge to distant poses from the original identity.

Furthermore, with each parameter being optimized independently, there is no information about the image that is being used, and no knowledge is shared when learning the parameters for each one of the images. We believe that learning all of the parameters at the same time sharing information should make the training much faster and more accurate. This is why we propose a new method using Convolutional Neural Networks to learn the camera parameters in the next section.

4.2. Pose Prediction

In this work, we explore whether learning the camera parameters can be learning the image parameters jointly for all images. Specifically, we try out *predicting* the parameters given in the training input image. We reason that as mentioned earlier, performing optimization using the information encoded about the position in each one of the images should be much more efficient than learning independent parameters for each one of the images. We propose using a module consisting of a Convolutional Neural Network (CNN) that given an input will return the camera parameters for the given image.

Using this module should fix both problems mentioned about the joint optimization training, learned information would now be reused for each image during training and we should avoid the issue with initialization of parameters. The reason why we believe the first problem is fixed is mentioned earlier and for the second problem, since the random initialization is performed over the network weights and not over the outputs of the neural network we believe we should avoid the aforementioned issue.

We are proposing to use a small CNN to learn the camera poses. More concretely we run our experiments with a CNN consisting of a convolutional layer with an output of 5 channels and a kernel size of 5 by 5. Followed by a ReLU activation function and a maxpool layer, with kernel size 3 by 3 and stride 3 by 3. Then a second convolutional layer with a single output channel, again kernel size 5 by 5, ReLU activation function, and same maxpool layer. Finally, we concatenate this with 2 fully connected linear layers with 500 neurons and outputting the 6 necessary features to recover the camera pose.

4.3. John Harvard Dataset

Looking at past research, most of the papers use the same small set of benchmark datasets to perform their experiments. We want to evaluate the robustness of these methods by testing them on a new dataset. We built a dataset of pictures of the John Harvard statue in the Harvard Yard for its 3D reconstruction.

This dataset has been created by taking multiple pictures of the statue using the camera of an iPhone 11 Pro. We did not have access to any special equipment to recover the camera positions for each picture which means that we do not have access to the camera position for any image. The pictures were taken trying to have the statue at the center of the image. We took multiple rounds of pictures on different levels. We maintained the camera around the same height and took multiple pictures at that level rotating around the statue by 150 degrees. Then we proceeded to do the same multiple times and change the height position of the camera. We assembled a total of 92 images. Multiple images from the dataset are presented in Figure 1.

Ultimately, this is how we envision NeFRs to work in the real world: reconstructing a 3d representation with noise input, not perfectly centered and aligned images, and unknown camera parameters. We will use this dataset to evaluate the practicality of the proposed approaches.

Figure 1. Six sample images from the John Harvard dataset

5. 360° Reconstruction with Curriculum Learning

One of the main challenges when jointly optimizing the camera position and 3d representation is the dependence on each other. As long as the camera poses are incorrect, learning the 3d representation is almost not possible. And to find the camera poses, at least somewhat accurate 3d representation is needed. [11] show that it is nevertheless possible to do this joint optimization for forward-facing scenes. That means that all images show the same side of the object. To get an intuition of why the joint optimization works in that case, consider the following simplified procedure:

1. We fix one input image. The NeRF learns an almost perfect representation of the 2d image.
2. We switch to a different input image. As both images show roughly the same content, we can hope the network tries to modify the camera position so that the two images align. This should be easier than relearning the entire 3d representation.

Now consider what happens if instead, the second image shows the object on the opposite side. Then, there is no way for the network to align the images and find suitable camera poses.

To avoid this problem for complex scenes, this work extends Wang et al.'s approach [11] to use curriculum learning [1]. This means we assume to have a rough ordering of pictures by side (i.e. first pictures from side A, then from side B, etc.) and gradually expand the range of viewing angles we train on.

This is a much weaker requirement than knowing the exact camera poses. As we show in subsection 4.3, when taking pictures with a regular camera, precise position and orientation value is simply not available. However, for the practitioner, it is easy to order the pictures in a way that fulfills some spacial locality.

An example of how this approach might look like is shown in Figure 2. There, we see the camera positions of the 360° Lego dataset that was used in [7]. We color-coded a possible batching into the three batches "top", "front", and "back". Then, we would first only train on the front batch, then on both the front and top batch and finally on all three. In practice, we would want to use a slightly finer batch splitting, for example into eight batches to ensure each new im-

age only adds a slightly new viewing angle. Of course, in reality, the camera positions are not available and the batching needs to be performed by the user.

Parameter Freezing According to our hypothesis, after finishing the training on a specific batch, the camera parameters should be roughly correct and not change much when we add more viewing angles. We, therefore, *almost freeze* the trained camera parameters before moving to the next batch. Here, "almost freezing" means we strongly reduce the learning rate for those parameters to 5% of the original learning rate.

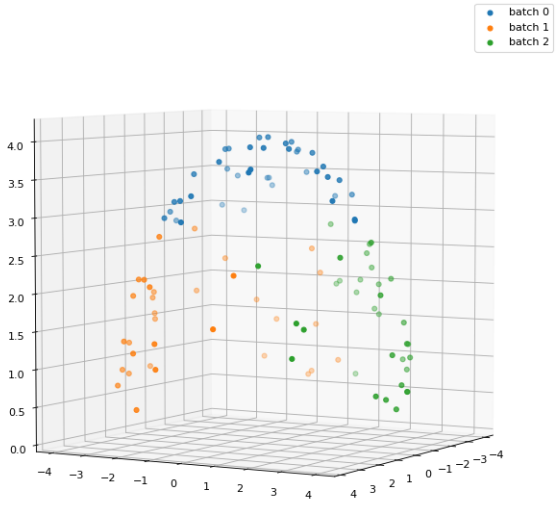

Figure 2. The camera positions for the 360° Lego dataset divided into three batches.

6. Experiments and Results

We evaluate our proposed methods in the following chapter and discuss the results. Note that due to limited computing resources, our models did not fully converge and our results might improve upon further training.

6.1. Pose Prediction on Forward Facing Scene

We designed an experiment to test our pose prediction method. We will compare it against the baseline presented in subsection 4.1 where the pose estimation is performed over joint optimization. Furthermore, we run these two models on the widely used as a benchmark LLFF dataset [6]. We run both models for 1000 epochs, and extract the PSNR to obtain its performance at each epoch.

The learning curves are presented in Figure 3. We observe that the joint optimization method does slightly better than our new proposed method. We believe that one of the reasons might be that this is a relatively easy dataset, and hopefully, our method would provide better results in more complex scenes. Where, by complex we mean, greater changes in camera positions in the training dataset. Nevertheless, we observe that the rendering of a new point of view from our method in Figure 4 is as visually appealing as for the fixed parameters.

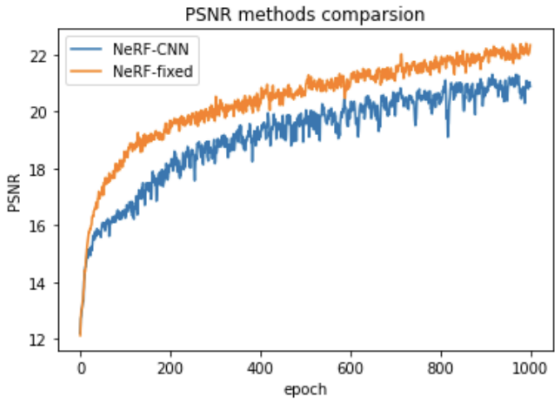

Figure 3. Learning curve of both models, joint optimization using fixed parameters and pose prediction using a CNN on the LLFF dataset.

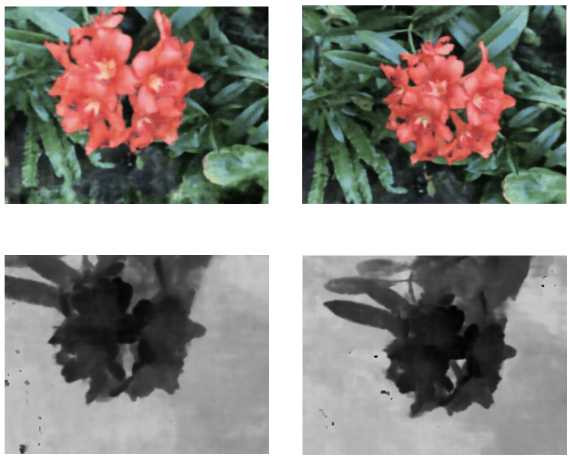

Figure 4. New point of view generated by our proposed method (right) and the benchmark (left) on an instance of the LLFF dataset. On top we have the rgb image and on the bottom we have a depth image.

6.2. Pose Prediction Robustness on John Harvard Dataset

Recall we wanted to test both of these methods in a less curated dataset to evaluate the robustness of these models and their suitability to be deployed in the real world. For this reason, we designed the John Harvard dataset, created from images taken with our mobile phones without any special equipment. We train both models on this new dataset

for 500 epochs and again extract the PSNR to obtain the performance at each epoch.

The results in Figure 6 are far from the ones we obtained with the flower in the previous experiment. The reason is that this dataset was not as carefully collected as the one in the previous experiment. We did not maintain always the same distance to the main object, the camera could be slightly tilted. Furthermore, the angle around the object was much larger in this dataset, making the learning much harder. However, looking at the learning curves presented in Figure 5, we observe that these did not plateau yet which makes us believe that the models did not converge yet and that it is possible to improve this further with more computing power. Furthermore, we believe that applying different techniques for example adding a perceptual loss term in the loss function during training could further improve the reconstruction, we tried this in the following section.

Nevertheless, we want to stress that the results in this experiment and the big difference in performance compared to the useful common datasets as LLFF is a red flag indicating that this method is still not ready for the being deployed in the real world because of its lack in robustness when data is not very carefully collected.

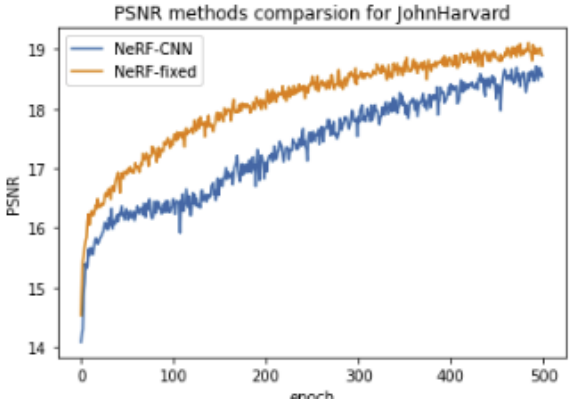

Figure 5. Learning curve of both models, joint optimization using fixed parameters and pose prediction using a CNN on the John Harvard Statue dataset.

6.2.1 Improving through Perceptual Loss

Adding a perceptual loss term in the loss function when optimizing Neural Networks for style transfer has proven to be very effective [4]. Seeing the results in the previous section is not as good as we would like, we tried whether adding a perceptual loss term in the loss function during training would help to reconstruct a better representation of the 3D scene. We use a VGG16 pre-trained model and obtain some latent representations of the expected output and the rendered output image. This latent representation is obtained

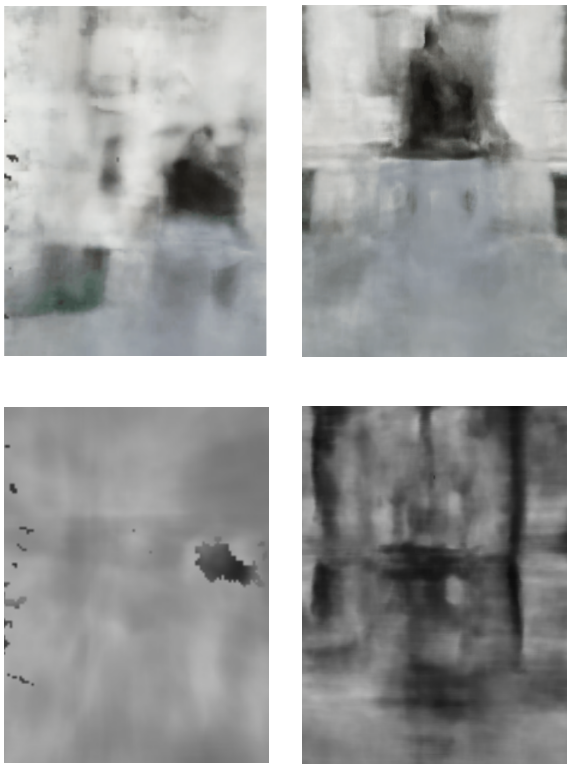

Figure 6. New point of view generated by our proposed method (right) and the benchmark (left) on the John Harvard Statue dataset. On top we have the rgb image and on the bottom we have a depth image.

by extracting the outputs of the pre-trained VGG16 when the image is passed in a middle layer. Then we obtain an MSE term that is added to the loss. Hence the resulting loss function is the following:

$$L = MSE(\text{image rendered, image expected}) + MSE(\text{VGG(image rendered), VGG(image expected)}) \quad (7)$$

We trained both models, our proposed method using the CNN and the previous method using fixed parameters with this modified loss. As in the previous experiment, these models are trained for 500 epochs.

In Figure 7 we observe that the PSNR curves reach a lower PSNR score than when the perceptual loss is not used. However, if we look at the results in Figure 8, these look much better than previously. We conclude that adding this perceptual loss really helped reconstruct the 3d rendering in a way that is much more visually appealing to the human eye.

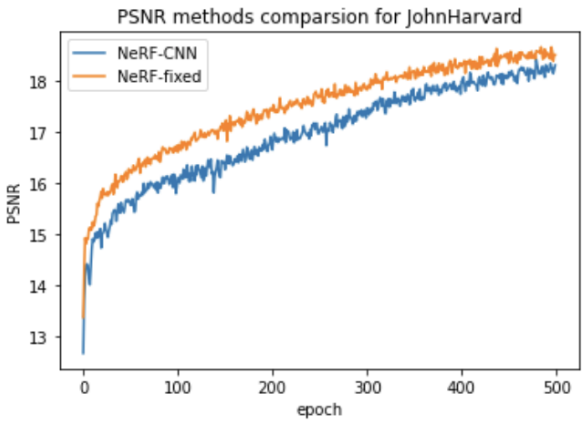

Figure 7. Learning curve of both models, joint optimization using fixed parameters and pose prediction using a CNN on the John Harvard Statue dataset and adding a perceptual loss term to the training loss.

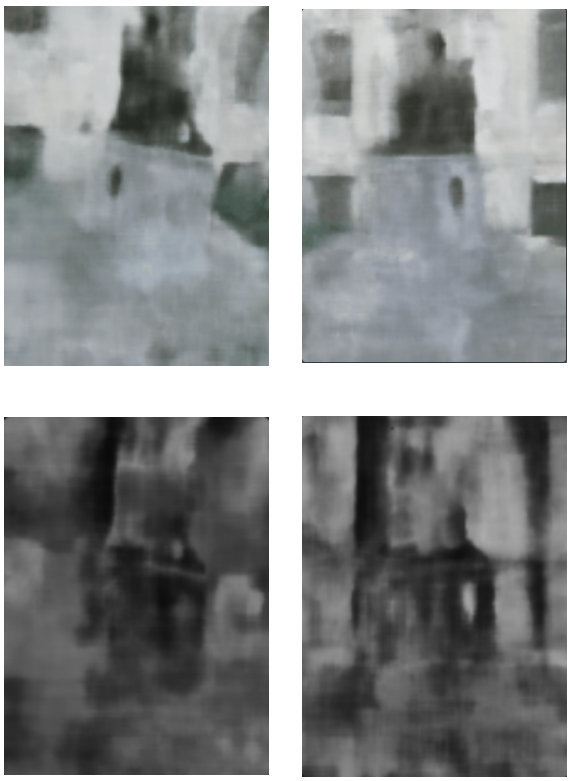

Figure 8. New point of view generated by our proposed method (right) and the benchmark (left) on the John Harvard Statue dataset adding the perceptual loss term to the training loss. On top we have the rgb image and on the bottom we have a depth image.

6.3. 360° reconstruction

As described in section 6.4 of [11], using their method to learn a 3d representation for the Lego case completely fails. We show their result after convergence in Figure 9.

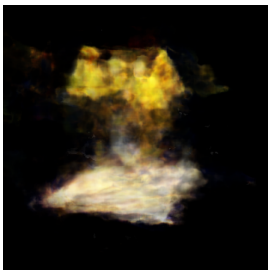

Figure 9. Reconstruction of the Lego dataset without camera parameters reported by [11].

To apply our curriculum learning approach, we split the dataset into 7 batches. One batch containing all images from the top (angle to the ground larger than 45°) and six batches containing renders from different sides. Figure 10 shows a sample of those batches. As we can see, pictures within a patch are quite homogeneous.

If we apply our curriculum learning approach to we receive the reconstructions shown in Figure 11. After training on only the first batch, the reconstruction is roughly correct (Figure 11a). However, as we add more viewing angles, the reconstruction becomes worse (Figure 11b). The same behavior can be observed in the PSNR score during training (Figure 12). In the first batch, the reconstruction score is high but then decreases as more viewpoints are added. While this result looks better than the baseline from Wang et al. [11] (Figure 9), we still fail to learn a viewpoint consistent 3d presentation.

7. Discussion

To conclude, in this project, we proposed a new way to train NeRFs without knowing the camera poses, using CNNs and training the whole system end-to-end. We compared its performance to another pre-existing method that was optimizing the camera pose independently for each image in the training set. We observed that our method performed slightly worse. However, we presented our reasons to believe that this should fix some present problems in the other methods such as restrictive initialization and lack of knowledge sharing between images while training. Hence we believe that with some further work, our method could beat the previous one.

Furthermore, we tested both methods in a newly generated dataset of pictures of the statue of John Harvard. We observed that these methods performed very poorly compared to when they are used in the more carefully curated

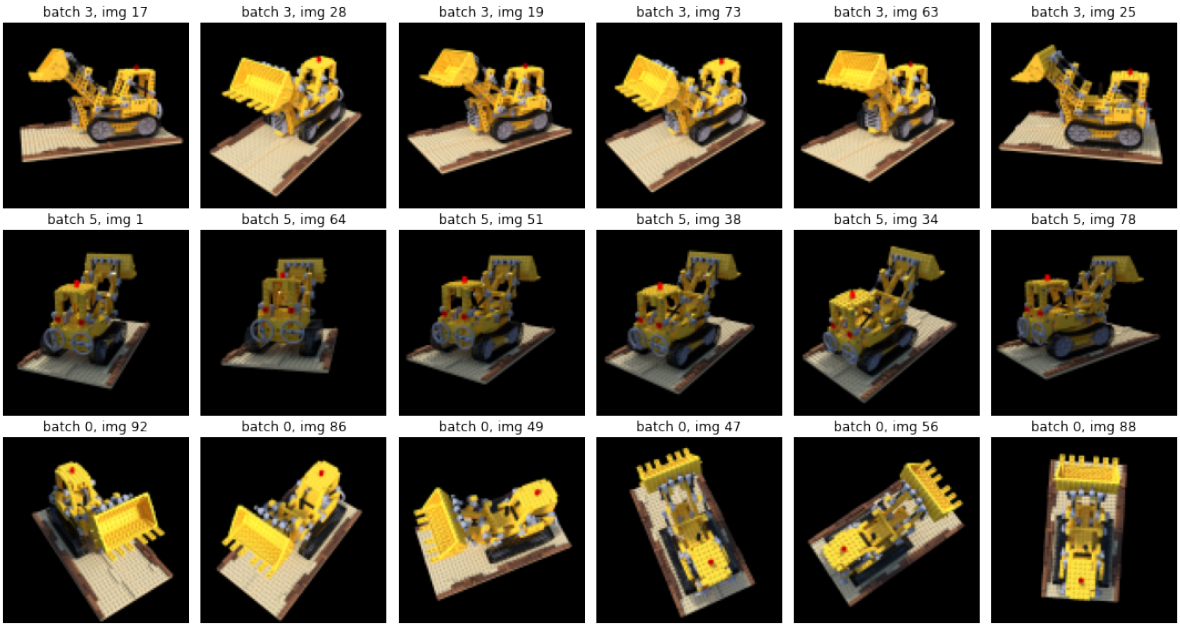

Figure 10. The 360° Lego dataset divided into seven batches (samples from three batches are shown).

(a) 1 batch (b) all batches

Figure 11. Our reconstruction after training on only one batch (a) and after training on all batches (b).

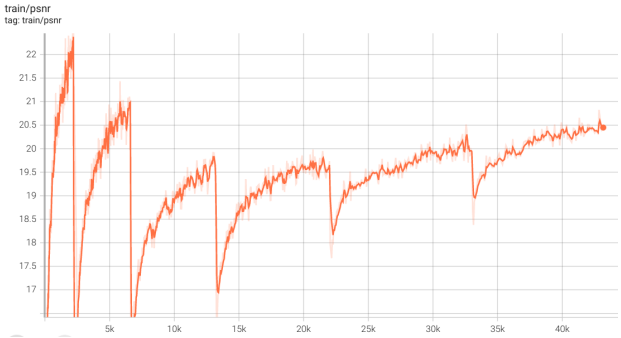

Figure 12. PSNR score over number training steps. We see a steep performance drop for each start of a new batch. The reconstruction performance from the first batch is never reached again.

dataset as the LLFF. We take this as a red flag indicating that there is further work to be done in order to be able to deploy these systems in the real world when datasets are not as carefully collected.

Finally, we implemented a curriculum learning strategy for 360° reconstruction without known camera positions. Our preliminary results show that our strategy generates slightly better reconstructions than the benchmark from Wang et al [11]. However, this improvement was not enough to obtain satisfactory results for the Lego 360° dataset.

8. Team Contributions

The project is joint work of Marcel Torne Villasevil and Florian Juengermann. Marcel was responsible for the camera parameter estimation with a CNN and the evaluation of the John Harvard dataset. Florian worked on the 360° reconstruction with curriculum learning.

References

[1] Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. Curriculum learning. In *Proceedings of the 26th annual international conference on machine learning*, pages 41–48, 2009. 4

[2] Andrew J Davison, Ian D Reid, Nicholas D Molton, and Olivier Stasse. Monoslam: Real-time single camera slam. *IEEE transactions on pattern analysis and machine intelligence*, 29(6):1052–1067, 2007. 2

[3] Jakob Engel, Thomas Schöps, and Daniel Cremers. Lsd-slam: Large-scale direct monocular slam. In *European con-*

ference on computer vision, pages 834–849. Springer, 2014. 2

[4] Justin Johnson, Alexandre Alahi, and Li Fei-Fei. Perceptual losses for real-time style transfer and super-resolution. *CoRR*, abs/1603.08155, 2016. 6

[5] Lingjie Liu, Jiatao Gu, Kyaw Zaw Lin, Tat-Seng Chua, and Christian Theobalt. Neural sparse voxel fields. *Advances in Neural Information Processing Systems*, 33:15651–15663, 2020. 2

[6] Ben Mildenhall, Pratul P. Srinivasan, Rodrigo Ortiz Cayon, Nima Khademi Kalantari, Ravi Ramamoorthi, Ren Ng, and Abhishek Kar. Local light field fusion: Practical view synthesis with prescriptive sampling guidelines. *CoRR*, abs/1905.00889, 2019. 5

[7] 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. In *The European Conference on Computer Vision (ECCV)*, 2020. 2, 4

[8] Raul Mur-Artal, Jose Maria Martinez Montiel, and Juan D Tardos. Orb-slam: a versatile and accurate monocular slam system. *IEEE transactions on robotics*, 31(5):1147–1163, 2015. 2

[9] Nasim Rahaman, Aristide Baratin, Devansh Arpit, Felix Draxler, Min Lin, Fred Hamprecht, Yoshua Bengio, and Aaron Courville. On the spectral bias of neural networks. In *International Conference on Machine Learning*, pages 5301–5310. PMLR, 2019. 2

[10] Matthew Tancik, Vincent Casser, Xinchen Yan, Sabeek Pradhan, Ben Mildenhall, Pratul Srinivasan, Jonathan T. Barron, and Henrik Kretzschmar. Block-NeRF: Scalable large scene neural view synthesis. *arXiv*, 2022. 1, 2

[11] Zirui Wang, Shangzhe Wu, Weidi Xie, Min Chen, and Victor Adrian Prisacariu. Nerf–: Neural radiance fields without known camera parameters. *arXiv preprint arXiv:2102.07064*, 2021. 1, 2, 3, 4, 7, 8

[12] Lin Yen-Chen, Pete Florence, Jonathan T Barron, Alberto Rodriguez, Phillip Isola, and Tsung-Yi Lin. inerf: Inverting neural radiance fields for pose estimation. In *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1323–1330. IEEE, 2021. 2

[13] Alex Yu, Ruilong Li, Matthew Tancik, Hao Li, Ren Ng, and Angjoo Kanazawa. Plenoctrees for real-time rendering of neural radiance fields. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 5752–5761, 2021. 2