

FastSR-NeRF: Improving NeRF Efficiency on Consumer Devices with A Simple Super-Resolution Pipeline

Chien-Yu Lin*
 University of Washington
 Seattle, WA, USA
 cyulin@cs.washington.edu

Qichen Fu, Thomas Merth, Karren Yang, Anurag Ranjan
 Apple, Inc.
 Cupertino, CA, USA
 {qfu22,tmerth,karren.yang,anuragr}@apple.com

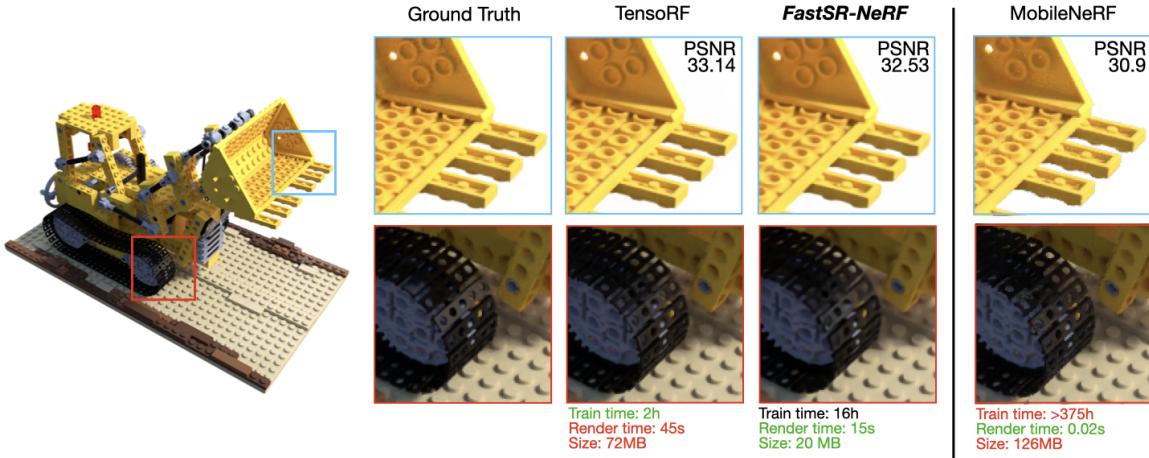


Figure 1. **Super-resolution (SR) can be used to improve neural rendering efficiency under a limited training budget.** Comparison of TensoRF, FastSR-NeRF (ours), and MobileNeRF [13] on a consumer-grade MacBook Air M2 laptop. FastSR-NeRF employs a straightforward SR pipeline (TensoRF+SR), which can enhance rendering times and compress the model size while incurring relatively low training overhead. While state-of-the-art mobile models such as MobileNeRF can render very quickly, they cannot be trained on consumer devices under a meaningful time budget.

Abstract

Super-resolution (SR) techniques have recently been proposed to upscale the outputs of neural radiance fields (NeRF) and generate high-quality images with enhanced inference speeds. However, existing NeRF+SR methods increase training overhead by using extra input features, loss functions, and/or expensive training procedures such as knowledge distillation. In this paper, we aim to leverage SR for efficiency gains without costly training or architectural changes. Specifically, we build a simple NeRF+SR pipeline that directly combines existing modules, and we propose a lightweight augmentation technique, random patch sampling, for training. Compared to existing NeRF+SR methods, our pipeline mitigates the SR computing overhead and can be trained up to 23× faster, making it feasible to run on

consumer devices such as the Apple MacBook. Experiments show our pipeline can upscale NeRF outputs by 2–4× while maintaining high quality, increasing inference speeds by up to 18× on an NVIDIA V100 GPU and 12.8× on an M1 Pro chip. We conclude that SR can be a simple but effective technique for improving the efficiency of NeRF models for consumer devices.

1. Introduction

Neural Radiance Field (NeRF) models [31] have become integral to many computer vision and computer graphics tasks, such as novel view synthesis [8, 29, 31], surface reconstruction [39, 45], camera pose estimation [41, 46] and 3D image generation [11, 28, 33]. Since the original NeRF model cannot render images efficiently, a large body of research [12, 17, 29, 32, 34, 37, 43] has been dedicated to address the rendering efficiency. Many of these

*Work done while interning at Apple.

works achieve impressive gains by decomposing and representing the 3D neural radiance field with explicit features [12, 13, 17, 21, 29, 32, 37]. However, these methods often require extended training times and/or specialized architectures and kernel support on high-end GPUs. For example, MobileNeRF is capable of fast rendering on mobile devices [13], but uses 8 server-class GPUs for training [6], which translates to over 15 days (>375 h) on a consumer-grade laptop (Figure 1, right). To improve the accessibility and personalized use of NeRFs, there is a need to explore efficient rendering techniques that can also be trained on consumer-grade devices.

In this paper, we introduce FastSR-NeRF, which demonstrates CNN-based super-resolution (SR) can be a simple, low-cost technique for improving the efficiency of NeRF models on consumer devices. The basic idea is to train a small NeRF model to generate lower-resolution scene features with 3D consistency, and a fast SR model to generate higher-resolution features. This combination reduces the number of pixels that need to be computed using the NeRF’s slow volume rendering process, increasing rendering speed. While SR techniques for neural rendering have been proposed by previous works, these methods either (i) involve specialized SR modules that use extra input features such as high-resolution reference images [23]; (ii) employ expensive training procedures such as distillation [10]; or (iii) are trained on tens of thousands of images within a generative modeling framework [11]. None of these methods can be feasibly trained on low-power, consumer-grade platforms. Whether it is possible to achieve high-quality results with SR under a limited training budget remains an open question.

Here, we address this question by exploring a simple NeRF+SR pipeline that directly combines existing modules. We hypothesize that the spatial inductive bias of CNN-based SR is sufficient to generate high-quality outputs for low upscaling ratios, even without extra input features or complex training procedures. To improve synthesis quality, we propose only a lightweight augmentation technique called ***random patch sampling***: rather than extract patches from an image grid for training the SR module as done in existing works [23, 40], we extract patches from random positions to increase the diversity of image patches seen by the SR module. Experiments across three datasets show, somewhat surprisingly, our simple NeRF+SR pipeline with low training overhead can achieve comparable quality and greater rendering efficiency than existing complex NeRF+SR pipelines. To summarize, the key results of our study are as follows:

- **SR can be a nearly “free” technique for improving neural rendering efficiency.** Our comprehensive experiments across three datasets show that applying SR to a NeRF model at 2-4 \times upscaling ratios, without any

complex training procedures or architectural modifications, can improve inference speeds by up to 35.7 \times on an NVIDIA V100 GPU and 12.8 \times on an M1 Pro chip, while maintaining peak signal-to-noise ratio (PSNR) in a 0.4-1.2 dB range. *Surprisingly, our simple pipeline can achieve comparable quality to recent and more complex SR techniques [23, 38], while being more efficient in training and inference.*

- **Random patch sampling is a crucial lightweight augmentation technique for NeRF+SR.** We propose random patch sampling, a lightweight augmentation technique. This augmentation improves the PSNR of the SR module by up to 0.89 dB for 2 \times upscaling and up to 1.44 dB for 4 \times upscaling compared to standard grid-based patch sampling, outperforming expensive distillation approaches [10] at a fraction of the time cost.
- **FastSR-NeRF is one of the few efficient methods that can be trained on a low-power device.** As shown in Figure 1, by utilizing a simple NeRF+SR pipeline, FastSR-NeRF can be trained on consumer devices such as a MacBook Air M2, whereas most other models and existing NeRF+SR pipelines fail to train with a meaningful time budget.

Overall, our analysis shows that SR can be a low-cost, plug-and-play strategy for improving the efficiency of neural rendering models under a limited training budget. Even a simple NeRF+SR pipeline can make neural rendering more efficient and accessible for those with low-power, consumer-grade hardware.

2. Background

2.1. Neural Radiance Fields

The NeRF model was first proposed in [31]. Given a position and view angle in a 3D scene, NeRF uses a large MLP network to map from the 5D input (3D coordinates plus 2D view angle) to an RGB and a density value. To render a 2D image, these MLP outputs are integrated along rays passing through each pixel using volume rendering. The MLP is optimized using gradient descent with respect to a photometric loss over a sparse set of scene-specific images. Due to its impressive results on static novel view synthesis, NeRF quickly propelled the state-of-the-art in many other fields, including 3D image generation [11, 28, 33], 3D scene editing [19] and landscape reconstruction [44]. However, the drawback of the original NeRF is that it takes a long time to render images due to the slow volume rendering process.

To address this issue, many works have since been proposed to improve NeRF’s rendering efficiency. One line of work [13, 18, 21, 47] maps the learned radiance field to explicit representations such as octree-based [47] or voxel-based [21] data structures. These methods achieve faster

rendering time at the cost of larger memory and training time requirements. Another line of work [16, 22, 26, 32] focuses on improving the sampling algorithm to reduce overall computation, which yields a modest amount of acceleration. An emerging series of works divides the radiance field into explicit voxels [29, 34, 37], or some efficient representation such as matrix decomposition [12], hash table [32], and triplane [9, 11, 35]. As these models usually make use of a mix of explicit representations and MLP, they are referred as hybrid NeRFs. Typically, hybrid NeRF models can highly accelerate training time and rendering speed, but need a relatively large model size. Among these efficient NeRF methods, there is a trend to develop customized GPU kernels to further accelerate the specialized operations designed for each method. Although using customized kernels can bring a major speedup, it limits the ability to deploy the model on different classes of GPUs and consumer-grade devices [10]. Orthogonal to these works, we explore the application of SR modules for improving NeRF rendering efficiency under a limited training budget. To maximize flexibility for running on low-end devices, we consider Python implementations that do not use customized GPU kernels.

2.2. Super Resolution with NeRFs

Super-resolution (SR) is a recent, still under-explored method for enhancing NeRF efficiency. EG3D [11] applies SR on top of volume rendering within a generative adversarial network for 3D faces. The SR module in their network is trained on tens of thousands of images, whereas we consider scene-specific optimization on much smaller datasets (*e.g.*, 20-200 images per scene). NeRF-SR [38] performs sub-pixel sampling to super-resolve outputs, but this requires more compute and thus a longer training time. MobileR2L [10] proposes a full CNN-based neural light field model and uses a SR model in its second stage, but their method is trained using an expensive distillation procedure. RefSR-NeRF [23] proposes a specialized SR module that uses high-resolution reference image as additional input, resulting in slower training and inference times. 4K-NeRF [40] synthesizes ultra high-resolution (4K) outputs using depth features as additional input and incorporates SR to achieve feasible inference times. Overall, these existing works all have high training overhead and are not meant to be optimized on lower-power consumer devices.

In our work, we approach SR techniques for neural rendering from a different perspective. Rather than develop a complex pipeline that pushes the limit of reconstruction quality on high-end GPUs, we ask what efficiency gains, if any, can be made from a simple NeRF+SR pipeline trained on consumer-grade hardware. Our experiments show that a simple NeRF+SR pipeline can achieve comparable quality to existing complex pipelines, while being lightweight enough to train on consumer-grade hardware.

3. Method

3.1. A Simple NeRF + SR Pipeline

As shown in Figure 2, our pipeline simply consists of a NeRF model concatenated with a CNN-based SR module. Given a ray $\mathbf{r} = \mathbf{o} + t\mathbf{d}$, where \mathbf{o} and \mathbf{d} are respectively the ray origin and direction, NeRF reconstructs the color $\widehat{C}(\mathbf{r})$ with volume rendering as follows:

$$\widehat{C}(\mathbf{r}) = \sum_{i=1}^N T_i \cdot (1 - \exp(-\sigma_i \delta_i)) \cdot c_i, \quad (1)$$

where N is the number of sampling points along the ray, δ_i is the distance between two point sampled at t_i and t_{i+1} , $T_i = \prod_{j=1}^{i-1} \exp(-\sigma_j \delta_j)$, and σ_i and c_i are the density and color respectively of a position \mathbf{x} in the 3D scene. In the original NeRF model, σ_i and c_i are computed by MLP networks \mathcal{F}_σ and \mathcal{F}_c given position and viewing direction. In our pipeline, we use a hybrid NeRF model [12] to achieve state-of-the-art quality with improved training time and rendering speed. To compute the density and color, we fetch radiance features from a grid-based decomposition \mathcal{G}_σ and \mathcal{G}_c , and then feed sampled features to MLP \mathcal{F}_σ and \mathcal{F}_c :

$$\sigma_i = \mathcal{F}_\sigma(\mathcal{G}_\sigma(\mathbf{x})), c_i = \mathcal{F}_c(\mathcal{G}_c(\mathbf{x}), \mathbf{d}) \quad (2)$$

Due to the more powerful discrete features, the MLPs \mathcal{F}_σ and \mathcal{F}_c in the hybrid NeRF are smaller than the ones used in vanilla NeRF.

To train and render with the full NeRF+SR pipeline, we sample \mathbf{r} from a patch of rays at low-resolution (LR) R_{LR}^P , perform volume rendering based on Equation 1, and upsample the output with SR module S to get the final high-resolution (HR) output H :

$$\forall \mathbf{r}_{\text{LR}} \in R_{\text{LR}}^P, H = S(\widehat{C}(\mathbf{r}_{\text{LR}}; \mathcal{F}; \mathcal{G})) \quad (3)$$

Note that the sampling here covers a contiguous 2D patch, which differs from the stochastic sampling of rays used for training standard NeRFs. The NeRF+SR pipeline is optimized in an end-to-end manner with respect to the loss computed over the high-resolution image patch:

$$\mathcal{L}_{\text{MSE}} = \sum_{\mathbf{r}_{\text{HR}}, \mathbf{r}_{\text{LR}}} \|C(\mathbf{r}_{\text{HR}}) - S(\widehat{C}(\mathbf{r}_{\text{LR}}; \mathcal{F}; \mathcal{G}))\|_2^2 \quad (4)$$

where C is the ground-truth color and \mathbf{r}_{HR} is the HR counterpart of \mathbf{r}_{LR} in R_{HR}^P . In practice, we use bilinear interpolation to downsample R_{HR}^P and get R_{LR}^P .

Comments on Efficiency. For a high-resolution NeRF model, the number of rays computed by Equation 1 will be R_{HR} , and will also require N in the order of thousands to millions to reconstruct details. In contrast, in a NeRF + SR

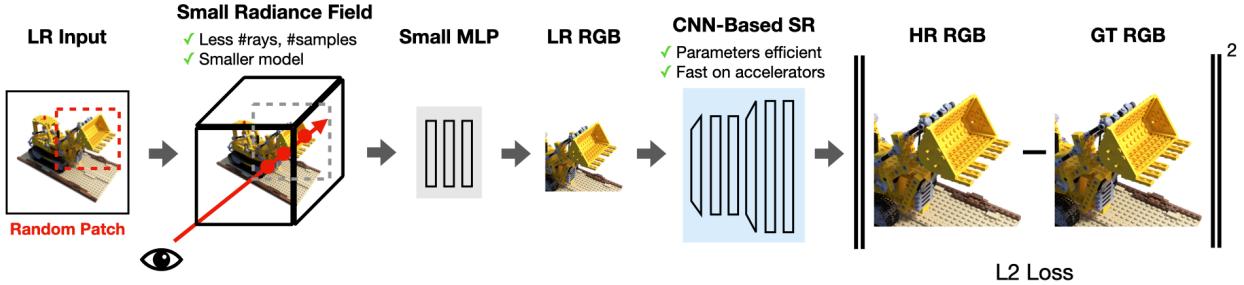


Figure 2. Overview of FastSR-NeRF. The SR module in our pipeline directly takes the RGB output from NeRF model, and therefore makes our pipeline easy to implement, model agnostic and flexible to run on different devices.

pipeline, the NeRF only needs output a low-resolution output, which reduces the number of rays to R_{LR} . N can also be reduced as there are fewer details in the lower-resolution image. In addition, we can reduce the grid and feature size of the hybrid NeRF to further improve NeRF efficiency, and still maintain the output quality at LR. As a result, we can greatly lower the computation overhead in Equation 1 and reduce the memory usage by letting NeRF output at LR.

The addition of the CNN-based SR module S does not present much of a computational bottleneck. With many years of progression on deep learning, the convolution operation is highly optimized and can efficiently run on modern commodity hardware such as GPUs [14], CPUs [5] and specialized accelerators [2, 7]. Furthermore, CNNs are parameter efficient by design [20, 25]. The memory savings from down-scaling the NeRF to LR are much more than the parameters overhead induced by the SR model, which reduces overall model size.

3.2. Random Patch Sampling

As discussed in Section 3.1, SR-based NeRF models need to be trained in a patch-style sampling instead of the tradition stochastic ray sampling. Previous works handle the patch sampling by dividing the rays of an image into equal-size patches following rigid grid lines which are determined by the given grid size. These ray patches are then shuffled and fed into the SR-based NeRF pipeline for supervision – we call this grid-based patch sampling. The problem of the grid-based patch sampling is that the sampling algorithm will cut the ray space strictly with a certain grid size. When the model gets trained on individual patches, there will be some variations that are never seen by the convolutional kernels as they sit between each grid boundaries.

To solve this issue, we propose **random patch sampling**. Instead of having a rigid grid lines and restricting sampling to the grid, we randomly sample a region in the ray space, and use that patch to train the model. In this way, we can still have a fixed patch size, but the content of each patch will be different regions of the input. After many it-

erations, the convolutional kernels in S will cover all of the patterns appeared in the training data and lead to a better training results.

In general, CNN-based models are prone to over-fitting and typically require large-scale datasets [15] to be trained. However, NeRF datasets usually only have tens to hundreds of training images, which is orders of magnitude smaller than a typical CNN pre-training dataset. Existing SR-based NeRF models tackle the overfitting issue by rendering extra data from a teacher model, or guiding training with additional features from high-resolution reference image or depth maps, but these significantly increase training overhead. Random patch sampling is a lightweight data augmentation technique that enables the convolutional kernels of the SR module to see more diversity in the training set. This crucial augmentation allows us to achieve high-quality results without the more complex architectures or training procedures of previous works. We provide ablation results of random patch sampling versus baselines in Section 4.5.

4. Evaluations

4.1. Datasets

We use the following three datasets for experiments.

NeRF Synthetic dataset. The NeRF Synthetic dataset was collected along with the origin NeRF [31] paper. It contains 8 different synthetic scenes with 360° degree views produced from the Blender [3] 3D computer graphics creation framework. Each scene in this dataset contains an object with complicated details. The object is placed in the middle of the 3D space and the background is white. For each scene, it has 100 training images and 200 testing images of the object from different views. The resolution of the collected images are 800×800.

NSVF Synthetic dataset The NSVF Synthetic dataset was released with the NSVF [29] paper. It has a similar setting as NeRF Synthetic dataset with gradually more complex geometry and lightening on the main object. The resolution is also at 800×800.

Method	NeRF-Synthetic					
	PSNR↑	SSIM↑	LPIPS↓	Train Time↓	Render Time(s)↓	Model↓ Size(MB)
NeRF [31]	31.01	0.947	0.081	~35h	20	5
TensoRF [12]	33.14	0.963	0.049	18m	1.4	71.8
FastSR-NeRF (2×)	32.53	0.961	0.052	1.5h	0.309	20
FastSR-NeRF (4×)	30.47	0.944	0.075	30m	0.077	13
FastSR-NeRF (8×)	27.27	0.902	0.142	16m	0.030	8
MobileR2L [10]	31.34	0.993	0.051	>35h	0.026 [‡]	8.3
NeRF-SR [38]	28.46	0.921	0.076	>35h	5.6	-

Method	NSVF-Synthetic					
	PSNR↑	SSIM↑	LPIPS↓	Train Time↓	Render Time(s)↓	Model↓ Size(MB)
NeRF [31]	30.81	0.952	-	~35h	~20	~5
TensoRF [12]	36.52	0.959	0.027	15m	1.4	74
FastSR-NeRF (2×)	35.39	0.979	0.032	1.5h	0.302	26
FastSR-NeRF (4×)	32.04	0.958	0.059	30m	0.075	12
FastSR-NeRF (8×)	27.93	0.911	0.119	16m	0.030	9

Method	LLFF					
	PSNR↑	SSIM↑	LPIPS↓	Train Time↓	Render Time(s)↓	Model↓ Size(MB)
NeRF [31]	26.5	0.811	0.250	~48h	33	5
TensoRF [12]	26.6	0.832	0.207	28m	5.9	188
FastSR-NeRF (2×)	26.20	0.822	0.241	2.5h	0.786	26
FastSR-NeRF (4×)	25.41	0.784	0.297	57m	0.165	15
FastSR-NeRF (8×)	21.30	0.584	0.475	15m	0.040	8
MobileR2L [10]	26.15	0.966	0.187	>48h	0.018 [‡]	8.3
NeRF-SR [38]	27.26	0.842	0.103	>48h	39.1	-
RefSR-NeRF [23]	26.23	0.874	0.243	-	8.5	38

Table 1. Quantitative and efficiency results on NeRF-Synthetic, NSVF-Synthetic and LLFF datasets. We compare the results of applying SR to the baseline NeRF model, TensoRF [12], and also list vanilla NeRF as a reference. For NeRF-Synthetic and LLFF, we also include the results of other SR-based models from their paper. The results are highlighted in red when there is a clear disadvantage of a method. [‡] The rendering time is for iPhone13, while other time is on GPUs.

LLFF dataset. LLFF dataset was collected along with the LLFF [30] paper. The scenes in this dataset were captured in the real world with forward-facing angle. It also has a major object placed roughly in the middle of each scene. However, different from the NeRF Synthetic dataset, the scenes in LLFF dataset have complex background depending on the captured environment. The original resolution of the collected images are 4032×3024 , and it also provide images at $4\times$ and $8\times$ lower resolution. Due to the practical usage, most of the NeRF works including us evaluate this dataset on 4x lower resolution at 1008×756 . Each scene in the LLFF dataset has 20 to 40 images, and 7/8 are used for training and 1/8 are used for testing.

4.2. Experiment Setup

We choose TensoRF [12] as our NeRF backbone as it achieves state-of-the-art results on both quality and efficiency, without requiring customized CUDA kernels, and therefore aligns with the goal of this paper. For our SR model, we chose EDSR [27] due to its accessible implementation and wide adoption in the computer vision community [4]. Although we choose TensoRF and EDSR as our NeRF and SR model, both of them can be replaced with other methods, as our pipeline is model agnostic. Since our SR module solely relies on the RGB output of the NeRF, we are able to leverage pretrained SR models. To train our pipeline, we first warm up the TensoRF model at LR using its default hyperparameters (inherited from the official implementation) for 5K iterations. After warming up,

we plug a pretrained EDSR model with desired SR ratio into our pipeline and start training end-to-end using random patch sampling. For the training hyperparameters, we fix the learning rate at 0.0001, patch size at 256 and 128 for SR-2 \times and SR-4 \times . We use Adam optimizer [24] and train the pipeline for 150 epochs. For each iteration in a epoch, we only feed one patch to the pipeline. We run our experiments on a machine that is equipped with a single NVIDIA V100 GPU with 16GB memory unless we specify the hardware platform.

4.3. Evaluation on Quality and Efficiency

Efficiency gains of utilizing SR. Here we evaluate the quality and efficiency gains of our simple NeRF+SR pipeline. We list peak signal-to-noise ratio (PSNR), structural similarity (SSIM) and perceptual similarity (LPIPS) [48] for quantitative quality measurements, and provide training time, rendering time and model size for efficiency evaluations. For LPIPS, we use VGG [36] as the backbone. The results can be found in Table 1.

As shown in Table 1, comparing to the backbone TensoRF [12] model, applying SR can generally maintain quality at the 2x ratio and enjoy efficiency benefits in rendering time and model size. For example, our pipeline with SR-2 \times only has a small 0.61dB, 1.13dB and 0.4dB PSNR drop and has near no loss on SSIM and LPIPS compared to the baseline model. Our pipeline at SR 2 \times even achieves a slight improvement on SSIM for NSVF-Synthetic. For efficiency, using 2x SR rate can improve rendering time by 4.5 \times , 4.6 \times and 7.5 \times and reducing model size by 3.6 \times , 2.8 \times , and 7.2 \times for NeRF-Synthetic, NSVF-Synthetic and LLFF respectively. For SR-4 \times , we observe a more notable quality loss to the baseline compared to SR-2 \times . However, it can still achieve qualified results such as over 30dB PSNR on synthetic datasets and just a small 1.19dB PSNR loss on LLFF dataset. At the mean time, with 4 \times SR rate, it can further improve the rendering time speedup to 18.2 \times , 18.6 \times and 35.7 \times , achieve model size reduction at 5.5 \times , 6.2 \times , and 12.5 \times for NeRF-Synthetic, NSVF-Synthetic, and LLFF. Furthermore, the training time for SR-4 \times is down to 30 min for synthetic datasets and 1hr for LLFF as the model run and converge faster at this rate. For SR-8 \times , although the efficiency is further improved, our pipeline can not maintain a good quality at this upscaling rate.

Comparing to existing SR-based NeRFs. We compare our simple pipeline to three existing SR-based model, which are MobileR2L [10], NeRF-SR [38] and RefSR-NeRF [23]. Notice that MobileR2L is light field based model and is not based on radiance field. However, they still utilize SR to enhance rendering speed so we include it for comparison.

Comparing to them, our simple pipeline with only lightweight techniques in training achieves a very clear advantage on the training time. At SR-2 \times , our pipeline can

Method	PSNR	M1 Pro		M2	
		Train Time	Render Time	Train Time	Render Time
TensoRF	33.14	2h	54s	2.5h	45.4s
FastSR-NeRF (2 \times)	32.53	22.5h	15.9s	16h	15.2s
FastSR-NeRF (4 \times)	30.47	15.5h	4.2s	13.5h	4s
MobileNeRF	30.9	>375h [†]	0.016s	>375h [†]	0.017s

Table 2. PSNR and time profiling of running vanilla TensoRF, FastSR-NeRF (ours) and MobileNeRF [13] on a MacBook Pro laptop with M1 Pro chip. [†] The training time is approximated for training the vanilla NeRF on M-series CPUs, which is only the first training stage of MobileNeRF.

be trained 23.3 \times and 19.2 \times faster than existing SR-based models on NeRF-Synthetic and LLFF, while achieving either on par or better quality.

Qualitative results. We show qualitative results and comparison on selected scenes from NeRF-Synthetic and LLFF datasets in Figure 3. As Figure 3 shows, with 2 \times upscaling rate, our pipeline can achieve on-par visual quality as the baseline TensoRF, while using bilinear interpolation at the same rate is not enough to get high fidelity results.

4.4. Training on Consumer Devices.

We run our pipeline on a MacBook Pro with M1 Pro chip and a MacBook Air with M2 chip to evaluate efficiency on consumer platforms. The training time, rendering time and PSNR on NeRF-Synthetic for our pipeline at SR rate 2 \times and 4 \times are listed in Table 2. We also list MobileNeRF’s [13] results for comparison.

As shown in Table 2, using SR can improve the rendering speed by up to 3.4 \times and 12.8 \times for 2 \times and 4 \times SR rate on the consumer-grade M-series CPUs. For MobileNeRF [13], although it can achieve a much faster rendering time from its specialized caching mechanism, it needs more than 15 days to be trained on the same device, which is difficult to make a meaningful NeRF application that run fully on a consumer-grade platform. In contrast, our pipeline, although can not achieve real-time rendering, it still significantly accelerates the rendering process with a reasonable training time (less than 1 day). Note that our experiments are run on CPUs because the current Apple Metal Performance Shader (MPS) [1] support in PyTorch can not fully run the operators needed in NeRF and SR on the MPS device. We expect our training and rendering speed to be faster once PyTorch has a better MPS operators support.

4.5. The Importance of Random Patch Sampling

We evaluate the effectiveness of random patch sampling, which we discussed in Section 3.2. To evaluate, we first establish our model pipeline as we explained in Section 3.1. We then keep all the settings of the pipeline the same but use different patch sampling algorithm to train our model. No-

Dataset	Method	PSNR	
		SR-2×	SR-4×
NeRF-Synthetic	Grid-based	31.84	29.28
	Random	32.53	30.47
NSVF-Synthetic	Grid-based	34.34	30.45
	Random	35.39	32.04
LLFF	Grid-based	26.2	24.94
	Random	26.04	25.41

Table 3. PSNR comparison on using **random patch sampling** versus **grid-based patch sampling**. We highlight the better results for the same SR ratio in **bold**.

tice that we fix the patch size and training for the same number of iterations, so the number of patches seen by the model is the same for both sampling methods. For grid-based patch sampling, we randomize the order of the patches fed into the model to ensure training stability. We report the averaged PSNR for SR ratio 2× and 4× of training our model with these two patch sampling algorithms in Table 3.

As Table 3 shows, we observe that random patch sampling consistently leads to a higher PSNR than grid-based patch sampling on synthetic datasets. The highest improvement appear on NSVF-Synthetic when the SR rate is 4×, where random patch sampling records a 1.59 PSNR increase over grid-base patch sampling.

On real-world forward facing LLFF dataset, the PSNR enhancement is less significant than the synthetic datasets. We observe a slight PSNR decrease (0.16dB) for SR-2× but a clear PSNR increase (0.47dB) for SR-4×. We hypothesize that this is because the real-world scenes typically contains greater complexity and finer-grained detail than the synthetic scenes. For example, in a synthetic scene, there is usually one major object and the space outside of the object is empty. Although, the object might has some difficult and fine-grained patterns, the model can focus on learning the patterns on the object. However, in a real world scene, although it usually still has a major object, the background is usually messy and contains a lot of small details. Therefore, using random patches on real world scenes such as LLFF dataset can not bring a big difference in terms of the total number of patterns converged in the patches. As a result, random patch sampling shows a greater improvement on synthetic datasets but still has the ability to enhance PSNR on real world scenes dataset when the SR rate is higher, e.g. 4×.

4.6. Ablation Study on Training Strategies.

In Section 4.5, we compare the results of training our NeRF + SR pipeline with grid-based or random patch sampling. However there are many more configurations possible. In this section, we compare the results of 1) using bilinear interpolation as the SR method, 2) use out-of-box pre-

Upsample Ratio	Upsample Method	Train Strategy	Train Time(m)	PSNR
2×	Bilinear	-	11	29.77
	EDSR	Pretrained	11	30.40
		Scratch	51	31.64
		FT-GridPatch	51	31.84
		FT-RandPatch	89	32.53
		Distillation	365	32.12
4×	Bilinear	-	3.5	26.67
	EDSR	Pretrained	3.5	27.62
		Scratch	19	29.03
		FT-GridPatch	19	29.28
		FT-RandPatch	30	30.47
		Distillation	166	29.94

Table 4. Training time and PSNR of different training strategies on NeRF Synthetic dataset. Pretrained EDSR model is downloaded from HuggingFace website. FT stands for finetuning the pretrained SR model. RandPatch signifies random patch sampling (other methods use grid-based patch sampling if not specified). All experiments are trained for 150 epochs, with the exception of distillation. For distillation, we generate 1k extra training image using a pretrained TensoRF as teacher, and train for 100 epochs.

trained SR model without finetuning, 3) training the NeRF and SR model both from scratch, 4) finetuning the pipeline with pretrained SR on the NeRF dataset with grid-based patch sampling, 5) same as (4) but uses random patch sampling, 6) use the distillation method proposed by [10] and train on a larger training set augmented by a teacher NeRF at HR.

We train the pipeline on NeRF-synthetic dataset and show the comparison in Table 4. Using bilinear interpolation as SR has the shortest training time (only requires warming up the NeRF backbone) but has a significant PSNR decrease. Directly using a pretrained EDSR model can also cut down the training time and has a better PSNR than bilinear. However, training on NeRF dataset still help it to achieve a better accuracy. Among those training methods, finetuning SR using random patch sampling achieves the best results while paying a little more training time (exclude distillation) due to the random sampling overhead. For training the pipeline, although we see a promising PSNR increase with 1K extra training images generated by a teacher TensoRF, the training time becomes much longer as we need to train the NeRF model at HR first and also need to render many HR images. Note that the PSNR of distillation might increase if we generate more training images, but the training time will be even longer. We do not further optimize our distillation procedure as it's not the focus in this paper. To sum up, using random patch sampling and finetuning a pretrained SR model gives us the best trade-off between time and quality under our pipeline setup.

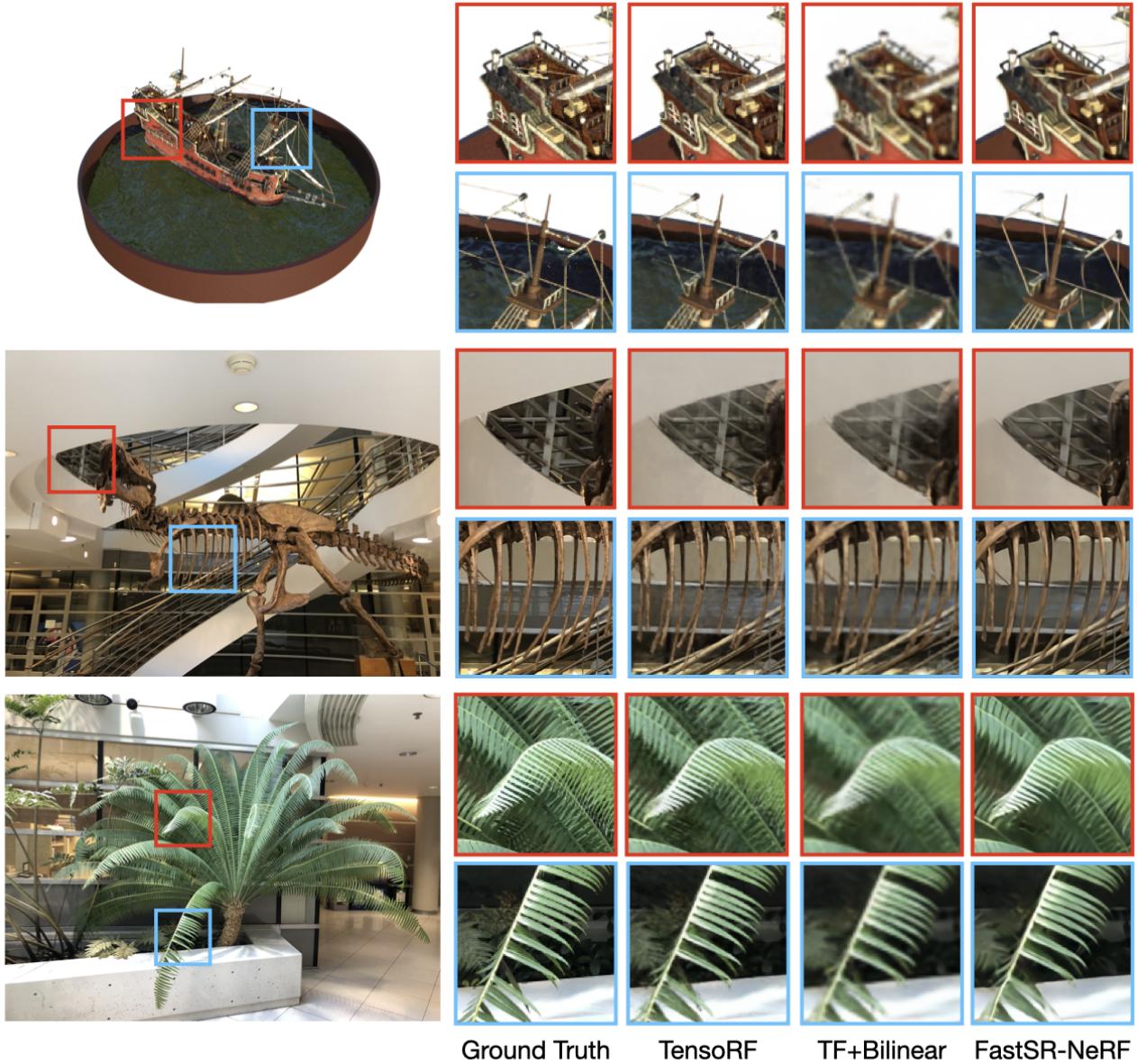


Figure 3. Qualitative results on a NeRF-Synthetic and LLFF. While TensoRF and TensoRF+Bilinear fail to recover some details, our pipeline successfully learn the details back with SR- $2\times$ rate.

5. Conclusion

In this work, we study the limit of SR-based NeRF model. We propose FastSR-NeRF and find a cohesive and simple NeRF + SR pipeline can actually achieve impressive quality while also being compute and memory efficient. The key result of this approach is that, although it's not the fastest nor the smallest model, it remains efficient for all of training time, rendering latency and model size. We achieve this by leveraging the lightweight technique called random patch sampling and pretrained SR model – both of these interventions can boost our pipeline's accuracy without in-

troducing prohibitive computational overhead. Our pure Python-based approach (without any customized GPU kernels) allows the whole training & inference pipeline to run on consumer-grade devices such as a laptop with a reasonable time. We believe this work and comprehensive analysis will help the development of an end-to-end NeRF application that can purely be deployed on personal devices for improved compute efficiency and user privacy.

References

- [1] Apple metal performance shader. <https://developer.apple.com/documentation/metalperformanceshaders/>. 6
- [2] Apple neural engine. <https://machinelearning.apple.com/research/neural-engine-transformers>. 4
- [3] The blender software. <https://www.blender.org/>. 4
- [4] Huggingface edsr. <https://huggingface.co/eugenesiow/edsr>. 5, 11
- [5] Intel onednn. <https://www.intel.com/content/www/us/en/developer/tools/oneapi/onednn.html>. 4
- [6] Mobilenerf official source code. <https://github.com/google-research/jax3d/tree/main/jax3d/projects/mobilenerf>. 2
- [7] Nvidia tensor core. <https://www.nvidia.com/en-us/data-center/tensor-cores/>. 4
- [8] Jonathan T. Barron, Ben Mildenhall, Matthew Tancik, Peter Hedman, Ricardo Martin-Brualla, and Pratul P. Srinivasan. Mip-nerf: A multiscale representation for anti-aliasing neural radiance fields. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 5855–5864, October 2021. 1, 13
- [9] Ang Cao and Justin Johnson. Hexplane: A fast representation for dynamic scenes. *CVPR*, 2023. 3
- [10] Junli Cao, Huan Wang, Pavlo Chemerys, Vladislav Shakhrai, Ju Hu, Yun Fu, Denys Makoviichuk, Sergey Tulyakov, and Jian Ren. Real-time neural light field on mobile devices. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8328–8337, 2023. 2, 3, 5, 6, 7, 13
- [11] Eric R. Chan, Connor Z. Lin, Matthew A. Chan, Koki Nagano, Boxiao Pan, Shalini De Mello, Orazio Gallo, Leonidas Guibas, Jonathan Tremblay, Sameh Khamis, Tero Karras, and Gordon Wetzstein. Efficient geometry-aware 3D generative adversarial networks. In *arXiv*, 2021. 1, 2, 3
- [12] Anpei Chen, Zexiang Xu, Andreas Geiger, Jingyi Yu, and Hao Su. Tensorf: Tensorial radiance fields. In *European Conference on Computer Vision (ECCV)*, 2022. 1, 2, 3, 5, 6, 13
- [13] Zhiqin Chen, Thomas Funkhouser, Peter Hedman, and Andrea Tagliasacchi. Mobilenerf: Exploiting the polygon rasterization pipeline for efficient neural field rendering on mobile architectures. In *The Conference on Computer Vision and Pattern Recognition (CVPR)*, 2023. 1, 2, 6, 13
- [14] Sharan Chetlur, Cliff Woolley, Philippe Vandermersch, Jonathan Cohen, John Tran, Bryan Catanzaro, and Evan Shelhamer. cudnn: Efficient primitives for deep learning. *CoRR*, abs/1410.0759, 2014. 4
- [15] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009. 4
- [16] Jiemin Fang, Lingxi Xie, Xinggang Wang, Xiaopeng Zhang, Wenyu Liu, and Qi Tian. Neusample: Neural sample field for efficient view synthesis. *arXiv:2111.15552*, 2021. 3
- [17] Sara Fridovich-Keil, Alex Yu, Matthew Tancik, Qinhong Chen, Benjamin Recht, and Angjoo Kanazawa. Plenoxels: Radiance fields without neural networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5501–5510, 2022. 1, 2, 13
- [18] Stephan J Garbin, Marek Kowalski, Matthew Johnson, Jamie Shotton, and Julien Valentin. Fastnerf: High-fidelity neural rendering at 200fps. *ICCV*, 2021. 2, 13
- [19] Ayaan Haque, Matthew Tancik, Alexei Efros, Aleksander Holynski, and Angjoo Kanazawa. Instruct-nerf2nerf: Editing 3d scenes with instructions. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2023. 2
- [20] Kaiming He, X. Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2015. 4
- [21] Peter Hedman, Pratul P. Srinivasan, Ben Mildenhall, Jonathan T. Barron, and Paul Debevec. Baking neural radiance fields for real-time view synthesis. *ICCV*, 2021. 2, 13
- [22] Tao Hu, Shu Liu, Yilun Chen, Tiancheng Shen, and Jiaya Jia. Efficientnerf efficient neural radiance fields. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 12902–12911, June 2022. 3, 13
- [23] Xudong Huang, Wei Li, Jie Hu, Hanting Chen, and Yunhe Wang. Refsr-nerf: Towards high fidelity and super resolution view synthesis. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 8244–8253, June 2023. 2, 3, 5, 6, 13
- [24] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations (ICLR)*, San Diego, CA, USA, 2015. 6
- [25] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012. 4
- [26] Rui long Li, Hang Gao, Matthew Tancik, and Angjoo Kanazawa. Nerface: Efficient sampling accelerates nerfs. *arXiv preprint arXiv:2305.04966*, 2023. 3
- [27] Bee Lim, Sanghyun Son, Heewon Kim, Seungjun Nah, and Kyoung Mu Lee. Enhanced deep residual networks for single image super-resolution. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, July 2017. 5, 11, 12
- [28] Chen-Hsuan Lin, Jun Gao, Luming Tang, Towaki Takikawa, Xiaohui Zeng, Xun Huang, Karsten Kreis, Sanja Fidler, Ming-Yu Liu, and Tsung-Yi Lin. Magic3d: High-resolution text-to-3d content creation. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2023. 1, 2

- [29] Lingjie Liu, Jiatao Gu, Kyaw Zaw Lin, Tat-Seng Chua, and Christian Theobalt. Neural sparse voxel fields. *NeurIPS*, 2020. 1, 2, 3, 4, 13
- [30] 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. *ACM Transactions on Graphics (TOG)*, 2019. 5
- [31] 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 *ECCV*, 2020. 1, 2, 4, 5, 13
- [32] Thomas Müller, Alex Evans, Christoph Schied, and Alexander Keller. Instant neural graphics primitives with a multiresolution hash encoding. *ACM Trans. Graph.*, 41(4):102:1–102:15, July 2022. 1, 2, 3, 13
- [33] Ben Poole, Ajay Jain, Jonathan T. Barron, and Ben Mildenhall. Dreamfusion: Text-to-3d using 2d diffusion. *arXiv*, 2022. 1, 2
- [34] Christian Reiser, Songyou Peng, Yiyi Liao, and Andreas Geiger. Kilonerf: Speeding up neural radiance fields with thousands of tiny mlps. In *International Conference on Computer Vision (ICCV)*, 2021. 1, 3, 13
- [35] Sara Fridovich-Keil and Giacomo Meanti, Frederik Rahbæk Warburg, Benjamin Recht, and Angjoo Kanazawa. K-planes: Explicit radiance fields in space, time, and appearance. In *CVPR*, 2023. 3
- [36] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *International Conference on Learning Representations*, 2015. 6
- [37] Cheng Sun, Min Sun, and Hwann-Tzong Chen. Direct voxel grid optimization: Super-fast convergence for radiance fields reconstruction. *CVPR*, 2022. 1, 2, 3, 13
- [38] Chen Wang, Xian Wu, Yuan-Chen Guo, Song-Hai Zhang, Yu-Wing Tai, and Shi-Min Hu. Nerf-sr: High quality neural radiance fields using supersampling. In *Proceedings of the 30th ACM International Conference on Multimedia, MM ’22*, page 6445–6454, New York, NY, USA, 2022. Association for Computing Machinery. 2, 3, 5, 6, 13
- [39] Peng Wang, Lingjie Liu, Yuan Liu, Christian Theobalt, Taku Komura, and Wenping Wang. Neus: Learning neural implicit surfaces by volume rendering for multi-view reconstruction. *NeurIPS*, 2021. 1
- [40] Zhongshu Wang, Lingzhi Li, Zhen Shen, Li Shen, and Liefeng Bo. 4k-nerf: High fidelity neural radiance fields at ultra high resolutions. *arXiv preprint arXiv:2212.04701*, 2022. 2, 3
- [41] Zirui Wang, Shangzhe Wu, Weidi Xie, Min Chen, and Victor Adrian Prisacariu. NeRF-: Neural radiance fields without known camera parameters. <https://arxiv.org/abs/2102.07064>, 2021. 1
- [42] Suttisak Wizadwongsu, Pakkapon Phongthawee, Jiraphon Yenphraphai, and Supasorn Suwajanakorn. Nex: Real-time view synthesis with neural basis expansion. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2021. 13
- [43] Zhijie Wu, Yuhe Jin, and Kwang Moo Yi. Neural fourier filter bank. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 14153–14163, June 2023. 1
- [44] Yuanbo Xiangli, Lining Xu, Xingang Pan, Nanxuan Zhao, Anyi Rao, Christian Theobalt, Bo Dai, and Dahua Lin. Bungeenerf: Progressive neural radiance field for extreme multi-scale scene rendering. In *The European Conference on Computer Vision (ECCV)*, 2022. 2
- [45] Lior Yariv, Jiatao Gu, Yoni Kasten, and Yaron Lipman. Volume rendering of neural implicit surfaces. *NeurIPS*, 2021. 1
- [46] Lin Yen-Chen, Pete Florence, Jonathan T. Barron, Alberto Rodriguez, Phillip Isola, and Tsung-Yi Lin. iNERF: Inverting neural radiance fields for pose estimation. <https://arxiv.org/abs/2012.05877>, 2020. 1
- [47] Alex Yu, Rui long Li, Matthew Tancik, Hao Li, Ren Ng, and Angjoo Kanazawa. PlenOctrees for real-time rendering of neural radiance fields. In *ICCV*, 2021. 2
- [48] Richard Zhang, Phillip Isola, Alexei A. Efros, Eli Shechtman, and Oliver Wang. The unreasonable effectiveness of deep features as a perceptual metric. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 586–595, 2018. 6

Appendices

A. Additional Ablation Study on Augmentations

In the present study, we introduce a technique of random patch sampling designed to improve the training efficacy of the NeRF+SR pipeline. In addition to this, we extend the same framework to incorporate additional data augmentations conventionally employed in Convolutional Neural Networks (CNNs), such as random rotation or perspective transformation applied to the sampled patches. Our central hypothesis posits that these slight image transformations could potentially generate novel patterns absent from the training set but pertinent to the 3D spatial context. Through these augmentations, the SR module is hypothesized to further generalize, thereby facilitating the recovery of lost details in unobserved perspectives.

Mathematically, the NeRF+SR pipeline involves the sampling of a patch in the low-resolution (LR) ray space R_{LR}^P and its high-resolution (HR) counterpart R_{HR}^P . A transformation \mathcal{A} is subsequently applied to these patches, yielding the transformed patches $R_{\text{LR}}^{P'}$ and $R_{\text{HR}}^{P'}$ as defined in Equation 5. These transformed patches are then integrated into Equation 3 for training the pipeline.

$$R_{\text{LR}}^{P'} = \mathcal{A}(R_{\text{LR}}^P), R_{\text{HR}}^{P'} = \mathcal{A}(R_{\text{HR}}^P) \quad (5)$$

To empirically validate our hypothesis, we implement two lightweight augmentations, random rotation and random horizontal flip, layered atop the random patch sampling technique. For synthetic datasets, the maximum rotation angle is set to 10 degrees, and the probability for a horizontal flip is set at 10%. Conversely, for the real-world scenes dataset LLFF, the maximum rotation angle is limited to 5 degrees, and the random horizontal flip is omitted since it's inappropriate with the forward-facing scenes.

The empirical results, presented in Table 5, reveal a marginal degradation in the output PSNR when using transformation-based augmentations as compared to utilizing random patch sampling exclusively. Consequently, random patch sampling remains as the most effective lightweight augmentation strategy for enhancing the NeRF+SR pipeline. We earmark the exploration of the effective utilization of transformation-based augmentations within the NeRF+SR pipeline for future research endeavors.

B. Additional Qualitative Results

We show additional qualitative results in Figure 4. Here we compare using different SR methods and different training procedures on the SR module. For the SR methods, we compare using bilinear interpolation and EDSR [27]. For

Dataset	Data Aug	2x	4x	8x
NeRF-Synthetic	Grid-Patch	31.84	29.28	26.02
	Rand-Patch	32.53	30.47	27.27
	RP+RRot+Hflip	32.22	30.26	27.26
NSVF-Synthetic	Grid-Patch	34.34	30.45	26.26
	Rand-Patch	35.39	32.04	27.93
	RP+RRot+Hflip	35.03	31.78	27.79
LLFF	Grid-Patch	26.2	24.94	21.68
	Rand-Patch	26.04	25.41	21.3
	RP+RRot	25.94	25.37	21.29

Table 5. PSNR of using different augmentation techniques for SR rate 2x, 4x and 8x. The output resolution is 800x800 for NeRF-Synthetic and NSVF-Synthetic, and 1008x756 for LLFF. Grid-Patch stands for grid-based patch sampling and Rand-Patch stands for random patch sampling. RRot stands for random rotation and Hflip stands for random horizontal flip. The best results in each SR rate and dataset are highlighted in bold.

different training procedures, we compare taking the pre-trained EDSR from [4], finetuning the EDSR model with grid-based patch sampling and finetuning the EDSR model with random patch sampling

As discerned from Figure 4, reliance on bilinear interpolation culminates in outputs characterized by a lack of sharpness, rendering them blurry. In contrast, utilization of a pretrained SR module yields images of greater clarity, albeit with some loss of intricate details. Subsequent finetuning facilitates the recovery of nuanced patterns, such as shadows. Remarkably, the deployment of our proposed random patch sampling methodology further enhances performance, as evidenced by improvements in the Peak Signal-to-Noise Ratio (PSNR) when compared to traditional grid-based patch sampling.

C. Additional Comparison with more NeRF models.

Beyond the data presented in Table 1, we extend our comparative analysis to encompass additional NeRF models specifically optimized for efficiency, incorporating both super-resolution (SR) based and non-SR-based approaches, as enumerated in Table 6. The empirical results delineated in Table 6 affirm that our proposed pipeline not only maintains high-quality output but also excels in terms of efficiency. This efficiency is observed across multiple metrics including training duration, rendering velocity, and model compactness, all achieved without necessitating specialized CUDA support on GPUs.

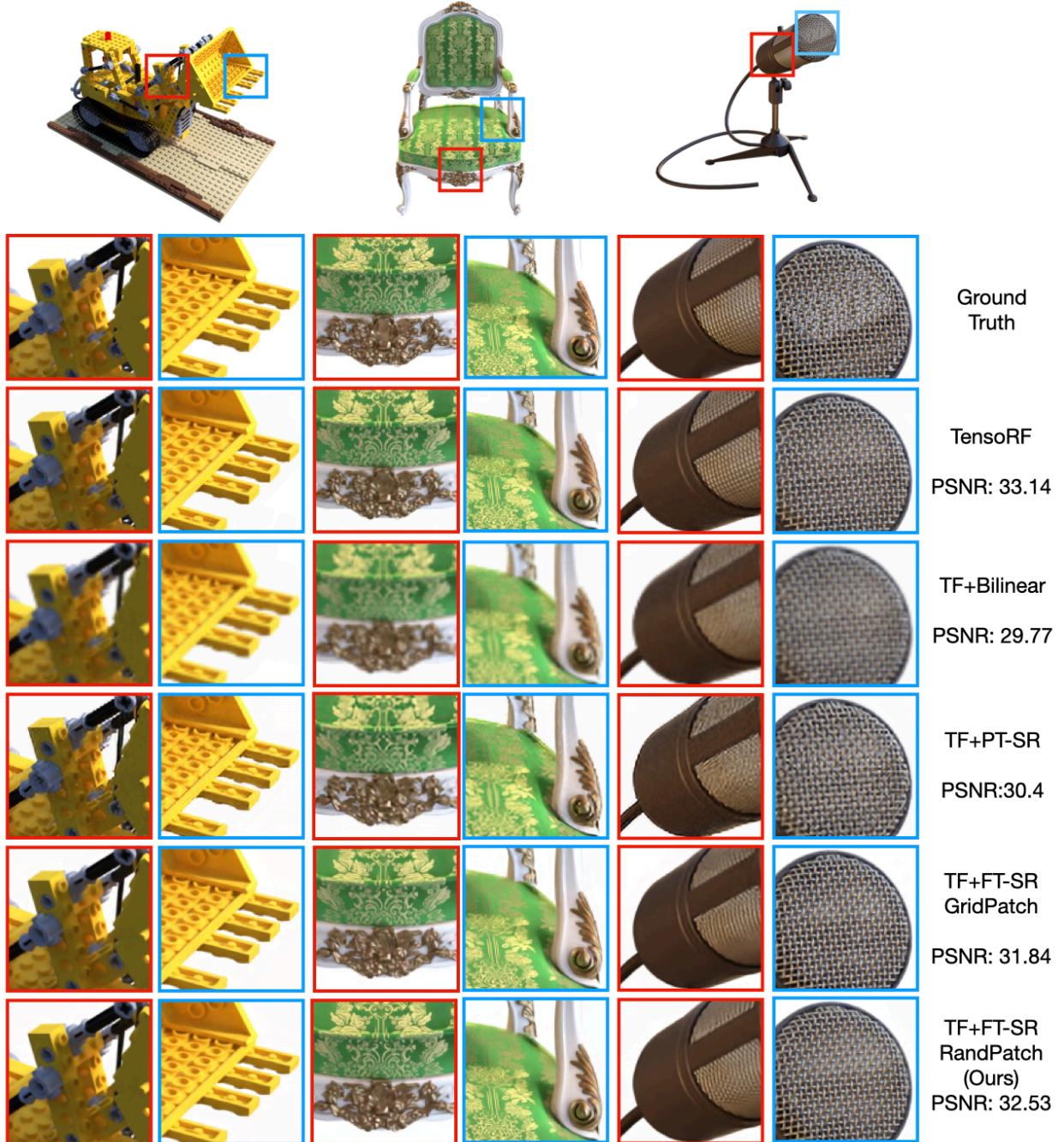


Figure 4. Qualitative results on lego, chair and mic scenes in NeRF-Synthetic. We show comparison on TensoRF at HR, and using bilinear, pretrained SR, finetuned SR with grid-based patch sampling and finetuned SR with random patch sampling to upsample output from TensoRF. The SR rate is $2\times$, and the SR module is EDSR [27]. We show the average PSNR on NeRF-Synthetic dataset of each method.

Method	PSNR↑	SSIM↑	LPIPS↓	Train Time	Render Time(s)	Model Size(MB)
NeRF [31]	31.01	0.947	0.081	~35h	20	5
MipNeRF [8]	33.09	0.961	0.043	~35h	-	-
NSVF [‡] [29]	31.74	0.953	0.047	>48h	3	-
KiloNeRF [†] [34]	31.00	0.950	0.030	>35h	0.026	-
SNeRG [21]	30.38	0.950	0.05	~35h	0.012	86.8
MobileNeRF ^{‡†} [13]	30.90	0.947	0.062	>35h	0.0013	125.8
Efficient-NeRF [22]	31.68	0.954	0.028	6h	0.004	~3000
TensoRF [12]	33.14	0.963	0.049	18m	1.4	71.8
DVGO [37]	31.95	0.958	0.053	14m	0.44	612
FastNeRF [18]	29.90	0.937	0.056	-	0.041	>7000
Plenoxel [†] [17]	31.71	0.958	0.049	11m	0.066	815
Instant-NGP [†] [32]	33.18	-	-	5m	0.016	16
MobileR2L [10]	31.34	0.993	0.051	>35h	-	8.3
NeRF-SR [38]	28.46	0.921	0.076	>35h	5.6	-
FastSR-NeRF (2×)	32.53	0.961	0.052	1.5h	0.309	20

(a) NeRF Synthetic Dataset results.

Method	PSNR↑	SSIM↑	LPIPS↓	Train Time	Render Time(s)	Model Size(MB)
NeRF [31]	30.81	0.952	-	~35h	~20	~5
NSVF [‡] [29]	35.13	0.979	-	>48h	~3	-
DVGO [37]	35.18	0.979	-	~20m	-	~600
TensoRF [12]	36.52	0.959	0.027	15m	1.4	74
FastSR-NeRF (2×)	<u>35.39</u>	0.979	<u>0.032</u>	1.5h	0.302	26

(b) NSVF Synthetic Dataset results.

Method	PSNR↑	SSIM↑	LPIPS↓	Train Time	Render Time(s)	Model Size(MB)
NeRF [31]	26.5	0.811	0.250	~48h	33	5
SNeRG [21]	25.63	0.818	0.183	~48h	0.036	310
NeX [42]	<u>27.26</u>	0.904	0.178	20h	0.0033	-
Efficient-NeRF [22]	27.39	<u>0.912</u>	0.082	4h	0.005	4300
TensoRF [12]	26.6	0.832	0.207	28m	5.9	188
MobileR2L [10]	26.15	0.966	0.187	>48h	-	8.3
NeRF-SR [38]	<u>27.26</u>	0.842	<u>0.103</u>	>48h	39.1	-
RefSR-NeRF [23]	26.23	0.874	0.243	-	8.5	38
FastSR-NeRF (2×)	26.20	0.822	0.241	2.5h	0.786	26

(c) LLFF Dataset results.

Table 6. Quality and efficiency results on NeRF-Synthetic, NSVF-Synthetic, and LLFF datasets. The tables are organized into three sections: implicit MLP-based NeRFs, efficient fully-explicit or hybrid NeRFs, and SR-based NeRFs, including our approach. Top performance in each quantitative metric is marked in bold, and the second best is underlined. Clear efficiency disadvantages are highlighted in red. Our tests run on a NVIDIA V100 GPU, while other results are their GPU results cited from respective papers when available. [‡] notes such method requires 8 high-end GPUs to train. [†] notes the method relies on customized CUDA kernels. Our method produces **excellent quantitative results that is on par or only slightly less than the state-of-the-art** cross all the benchmarks. Our method further achieves **great efficiency results across training time, rendering speed and model size** without the need of customized CUDA kernels support, which is favorable for inexpensive consumer-grade devices.