Sobolev Training for Implicit Neural Representations with Approximated Image Derivatives: Supplementary Material

Wentao Yuan[1,2] Qingtian Zhu[1] Xiangyue Liu[1] Yikang Ding[1]
Haotian Zhang[1⋆] Chi Zhang[1]

[1]Megvii Research [2]Peking University

1 Additional Details & Results

1.1 Image Regression

For image regression task, we set the angular velocity of the sinusoidal function to 30 and initialize the weights of MLPs following [10]. We provide other results of Set 5 dataset [3] in Fig. 1. We also report the memory consumption as well training time required in Tab. 1. The proposed training paradigm requires more memory and time at the training phase for additional computation and storage of derivatives but it is worth noting that the time, as well as the memory consumption at the inference phase, is the same with or without the derivative supervision.

1.2 Inverse Rendering

For inverse rendering task, we set the angular velocity of the sinusoidal function to 1 and initialize the weights of MLPs according to the method described in [6]. LLFF dataset [8,9] consists of 8 scenes captured with a handheld cellphone, captured with 20 to 62 images. We follow [9] to hold out $\frac{1}{8}$ of images for the evaluation set. All of the training images are 756×1008, but the dataset also provides the raw cellphone images of 3024×4032, which will be used to evaluate high resolution rendering results in Sec. 4.

We provide other results of LLFF dataset [8,9] in Fig. 2. Tab. 2 reports the training-time memory consumption and time. Similar to image regression, the additional cost is only at training-time.

2 Audio Regression

As mentioned in the discussions of the main paper, the training paradigm we proposed can further generalize to more tasks, as long as the tasks satisfy the formulation in Sec. 3.1 of the main paper. Here we follow [10] to conduct the task of audio signal representation.

This work is done by the first four authors as interns at Megvii Research.

⋆ Corresponding author (zhanghaotian@megvii.com).

Table 1. GPU memory consumption at training phase and training time of 1000 epochs in image regression task of *Baby*. P.E. stands for positional encoding; S.T. stands for Sobolev training.

Method	Memory(MB)	Time(s)
ReLU+P.E. [9]	130	4.678
ReLU+P.E.+S.T.	316	20.548
ReLU+P.E.+S.T.*	401	14.081
SIREN [10]	176	3.977
SIREN+S.T.	753	22.963
SIREN+S.T.*	704	16.556

* The new PyTorch 1.11.0 supports computation of per-sample derivatives, so we also report the statistics obtained.

Table 2. GPU memory consumption at training phase and training time of 1000 iterations in inverse rendering task of *Fern*.

Method	Memory(MB)	Time(s)
ReLU+P.E.	228	87.264
ReLU+P.E.+S.T.	912	190.434
SIREN+P.E.	348	86.642
SIREN+P.E.+S.T.	1827	204.748

Implementation

Dataset Following [10], we use two different audio signals, one for music and one for speech. For music data, we use the first 7 seconds from Bach's Cello Suite No.1 (*Bach*)*, and for the speech, we use stock audio of a male actor counting from 0 to 9 (*Counting*)**. Both audio signals share a sampling rate of $44100Hz$, and in total there are 308207 samples in *Bach* and 537936 samples in *Counting*. We normalize signal values to the range of $[-1, 1]$. Same as other tasks, we separate all samples within an audio clip into two groups, namely training samples and evaluation samples, by performing nearest-neighbor downscaling by a factor of 5. The approximated derivatives are obtained by two-sided differences method, similar to the vanilla derivative filter of the task on images.

Network Architecture We implement a fully-connected MLP with 4 activated linear layers with 256 hidden units and 1 output linear layer. We set the angular velocity of the sinusoidal function to 30 and follow the initialization strategy in [10] to initialize the weights of MLPs.

* Audio file available at https://www.yourclassical.org/episode/2017/04/04/daily-download-js-bach--cello-suite-no-1-prelude.

** Audio file available at http://soundbible.com/2008-0-9-Male-Vocalized.html.

Table 3. Quantitative PSNR results on the task of audio regression on *Bach* and *Counting*. 1/5 samples are used for training, and rest samples are used for evaluation. When trained with additional supervision on derivatives, PSNR is significantly improved.

Method	**Mean**	*Bach*	*Counting*
SIREN [10]	35.89	41.50	30.28
SIREN+S.T.	**41.23**	**48.89**	**33.57**

Training & Evaluation At the stage of training, we use the entire set of training samples to train the network as a batch with Adam optimizer, at a learning rate of 5×10^{-5}. For evaluation, we measure PSNR only on the evaluation samples.

Results The quantitative results in PSNR of regressing these two audio clips are shown in Tab. 3. Sobolev training significantly improves the regression quality of both audio clips. We further illustrate the visualized comparisons respectively in Fig. 3 and Fig. 4, showing the regression error of the derivative-trained network is smaller.

As our results show ReLU-activated networks can not fit audio signals well, which is consistent with the result of [10], we only report the results of Sine-activated networks.

3 Different Activation Functions

Recent ReLU-based MLPs normally have poor convergence property under derivative supervision, we mainly investigate the performance difference between ReLU and periodic activation function, i.e., sine, in the main paper. In this part, an experiment is designed to study the convergence properties of derivatives with different activation functions. The scope of the investigation is shown in Tab. 4.

Dataset We convert the RGB image *Bird* of Set 5 [3] to grayscale as our dataset. The original image shape is $288 \times 288 \times 1$, we use the Sobel operator to get approximate image partial derivatives, resulting the derivative image with shape $288 \times 288 \times 2$, w.r.t. u and v respectively. We perform nearest-neighbor downsampling on the original image and derivative image by a factor of 4.

Network Architecture The network architecture is the same as the image regression task in the main paper, except for the channel number of the last linear layer is 1 instead of 3. The angular velocity of the sinusoidal function and weight initialization method are both the same as image regression task. Taking the conclusion in [11,13] into consideration, we do two parallel experiments with and without positional encoding for all activation functions except for sine.

Training & Evaluation We use the downsampled 1/16 pixels as our training data and the remaining pixels as evaluation samples. The network is optimized for 10k iterations at a learning rate of 10^{-4}.

Table 4. Different activation functions and their definitions and weight initialization

Activation	Definition	Derivative	Initialization
ReLU [1]	$f(x) = \begin{cases} x, & x > 0 \\ 0, & x \leq 0 \end{cases}$	$f'(x) = \begin{cases} 1, & x > 0 \\ 0, & x < 0 \end{cases}$	Kaiming normal[6]
ELU [4]	$f(x) = \begin{cases} x, & x > 0 \\ \alpha(e^x - 1), & x \leq 0 \end{cases}$	$f'(x) = \begin{cases} 1, & x > 0 \\ \alpha e^x, & x < 0 \end{cases}$	Normal
SELU [7]	$f(x) = \begin{cases} \lambda x, & x > 0 \\ \lambda\alpha(e^x - 1), & x \leq 0 \end{cases}$	$f'(x) = \begin{cases} \lambda, & x > 0 \\ \lambda\alpha e^x & x \leq 0 \end{cases}$	Normal
Sigmoid	$f(x) = \frac{1}{1+e^{-x}}$	$f'(x) = f(x)(1 - f(x))$	Xavier normal [5]
Softplus	$f(x) = ln(1 + e^x)$	$f'(x) = \frac{1}{1+e^{-x}}$	Kaiming normal[6]
Tanh	$f(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$	$f'(x) = 1 - (f(x))^2$	Xavier normal [5]
Sine [10]	$f(x) = sin(x)$	$f'(x) = cos(x)$	Specific uniform [10]

Table 5. Quantitative comparisons of different activation functions on the task of (grayscale) image regression.

Activation	**PSNR↑**	**SSIM↑**
ReLU	25.06	0.711
ReLU P.E.	29.36	0.856
ELU	21.13	0.537
ELU P.E.	29.76	0.879
SELU	20.20	0.433
SELU P.E.	28.55	0.837
Sigmoid	18.06	0.408
Sigmoid P.E.	24.27	0.638
Softplus	18.38	0.417
Softplus P.E.	24.24	0.639
Tanh	23.27	0.635
Tanh P.E.	31.14	0.902
Sine	**33.13**	**0.960**

Results Fig. 5 and Fig. 6 respectively show the convergence curve of derivative loss with and without positional encoding. Sine demonstrates its great power in fitting the network's derivative compared with other activation functions. It is worth noting that positional encoding is helpful to improve the MLPs' ability of approximating derivatives when applied with other activation functions. The corresponding quantitative results are shown in Tab. 5.

4 Precision of Approximate Image Derivatives

In all aforementioned experiments, the partial derivatives we leverage for supervision are obtained by applying Sobel filters on the images before downsampling and for training, we perform nearest-neighbor downsampling for both image values and approximated derivatives. This is generally reasonable since in real

applications, the original images are usually of very high resolution, such as LLFF dataset [8,9], whose original resolution is 3024×4032. Without getting deteriorated rendering results, we would like the training data of INRs to be as few as possible for faster training and a direct solution is to downscale the images. As our results in the main paper show, by keeping the derivatives obtained at a high resolution, the problem of downgraded performance will get alleviated.

We also conduct experiments of image regression and inverse rendering where derivatives are obtained from the downsampled images, abandoning the data dependence of the raw data before downsampling, which is to say, we will use all samples in hand for training.

4.1 Image Regression

The dataset we used consists of 5 images of 1356×2040, which are generated by uniformly sampling images from DIV2K [2] validation set. We downsample the original images by nearest-neighbor interpolation with a factor 4 and use Sobel filters to calculate the approximate derivatives on downsampled images. The network architecture and training & evaluation settings are consistent with the image regression task of the main paper.

Results The quantitative results are shown in Tab. 6. From Tab. 6, we can see training with derivatives from downsampled images still gives a great performance improvement than value-based training paradigm.

4.2 Inverse Rendering

The experiment settings are different from the main paper. Here we do not perform downsampling to images; instead, we use images of 756×1008 for training and render novel views of 756×1008 and 3024×4032. The approximated image derivatives are calculated from images of 756×1008. As is mentioned, LLFF dataset [8] provides raw images of 3024×4032 so we can evaluate rendering results at a higher resolution.

Results The quantitative results on rendering views of different resolutions are shown in Tab. 7. As can be proven by the results, we can still get a slight performance improvement when training with approximated derivatives from downsampled training images. Also, the performance gap is enlarged, when the rendering views' resolutions are higher, indicating the great generalizability of Sobolev trained MLPs.

The difference between the two sources of image derivatives is basically the precision of approximated derivatives. As shown in the results of both tasks, the precision of approximated derivatives does not affect much. It is using the derivatives for supervision that matters.

Table 6. Quantitative results on the task of image regression on DIV2K validation set.

Method	**Mean**	DIV2K Validation Set				
		0820	*0840*	*0860*	*0880*	*0900*
PSNR ↑						
ReLU+P.E. [9]	23.34	20.44	25.75	18.62	29.36	22.51
ReLU+P.E.+S.T.	23.91	20.90	26.30	19.17	30.00	23.18
SIREN [10]	22.69	19.70	24.87	17.23	29.21	22.45
SIREN+S.T.	**24.44**	**21.58**	**26.87**	**19.35**	**30.64**	**23.74**
SSIM ↑						
ReLU+P.E. [9]	0.577	0.497	0.651	0.371	0.817	0.547
ReLU+P.E.+S.T.	0.625	0.559	0.692	0.401	0.843	0.629
SIREN [10]	0.594	0.508	0.650	0.356	0.842	0.616
SIREN+S.T.	**0.703**	**0.663**	**0.754**	**0.488**	**0.875**	**0.733**

Table 7. Quantitative results of using different paradigms when training with images of resolution 756×1008 and rendering novel views of 756×1008 and 3024×4032.

Method	Render $H \times W$	**Mean**	*Fern*	*Flower*	*Fortress*	*Horns*	*Leaves*	*Orchids*	*Room*	*T-Rex*
PSNR ↑										
ReLU+P.E. [9]	756×1008	24.69	24.26	25.92	28.58	25.32	20.39	**19.991**	**28.54**	24.52
	3024×4032	23.12	22.24	24.93	27.22	23.66	18.90	19.258	**26.24**	22.54
SIREN+P.E.+S.T.	756×1008	**24.72**	**24.33**	**26.06**	**28.66**	**25.40**	**20.65**	19.985	28.03	**24.68**
	3024×4032	**23.22**	**22.32**	**25.07**	**27.26**	**23.72**	**19.10**	**19.261**	26.14	**22.88**
SSIM ↑										
ReLU+P.E. [9]	756×1008	0.755	0.744	0.788	0.792	0.746	0.646	0.606	**0.900**	0.816
	3024×4032	0.723	0.714	0.773	0.819	0.703	0.570	0.616	**0.860**	0.732
SIREN+P.E.+S.T.	756×1008	**0.767**	**0.755**	**0.807**	**0.819**	**0.760**	**0.666**	**0.614**	0.899	**0.818**
	3024×4032	**0.729**	**0.717**	**0.781**	**0.830**	**0.705**	**0.581**	**0.622**	0.859	**0.736**

5 Use of Existing Assets

Some codes of image regression task and audio regression task are borrowed from SIREN [10]. The implementation of inverse rendering task are based on NeRF-PyTorch [12], which is a PyTorch version of original NeRF [9].

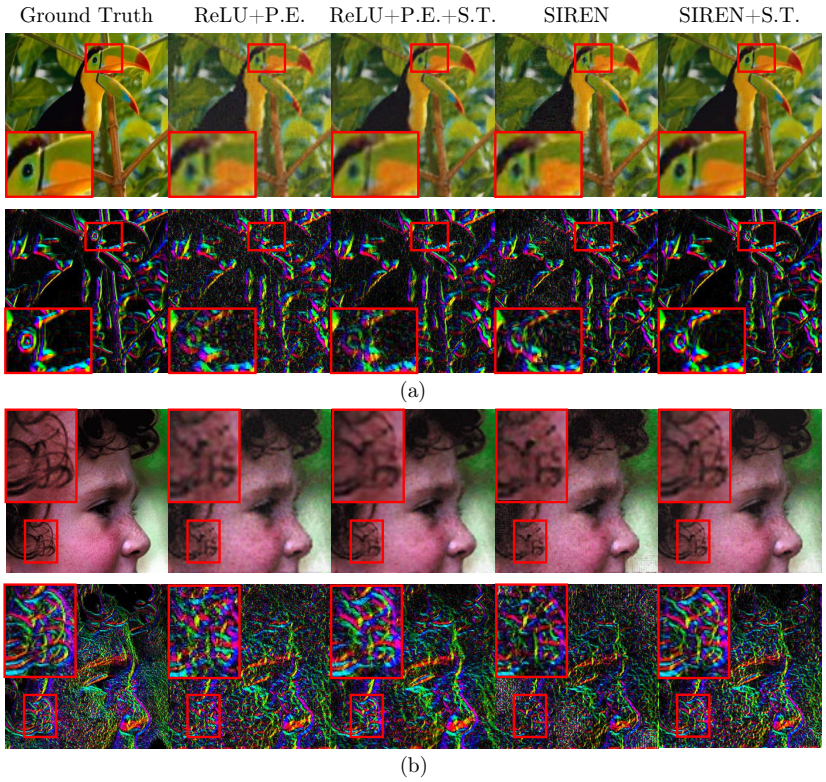


Fig. 1. Additional results of image regression. *Bird* (a), *Head* (b).

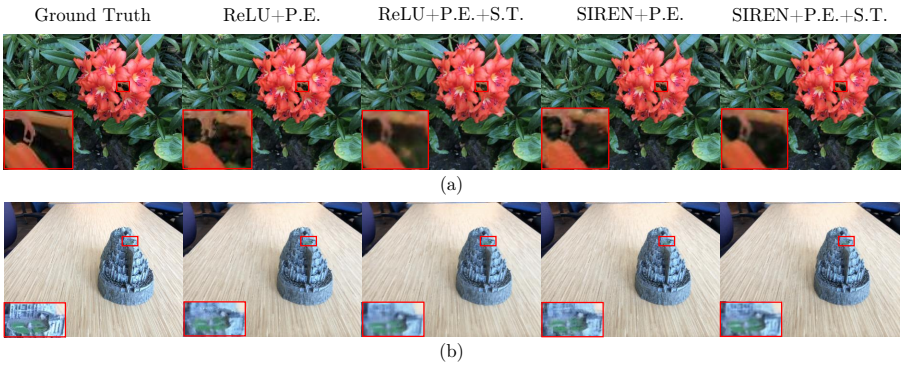
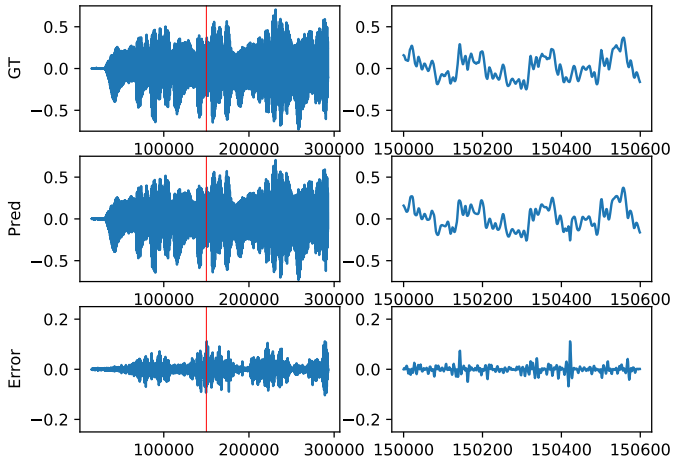
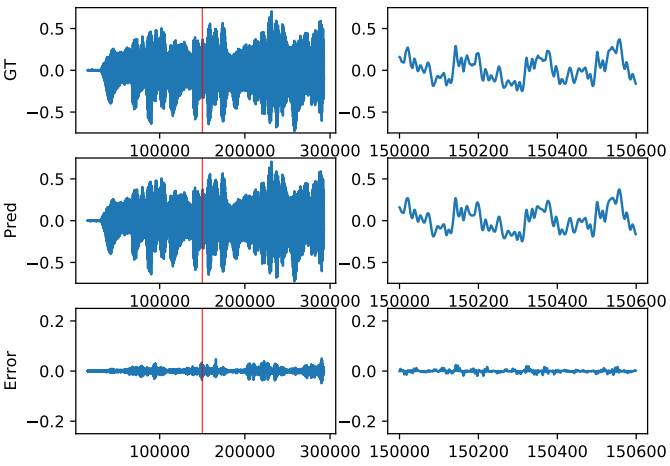


Fig. 2. Additional results of inverse rendering. *Flower* (a), *Fortress* (b).

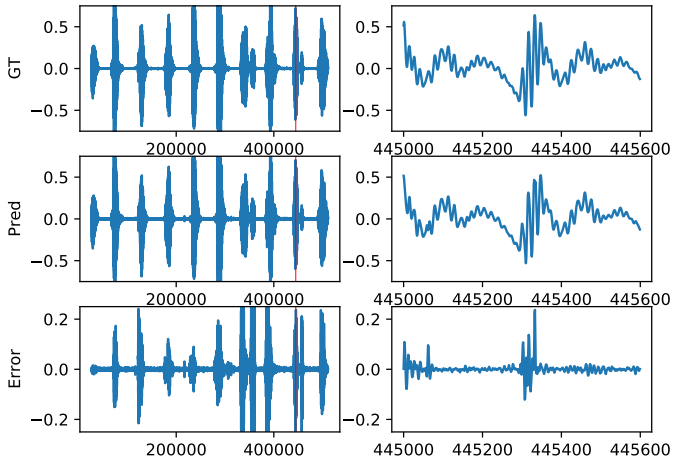


(a) SIREN [10]

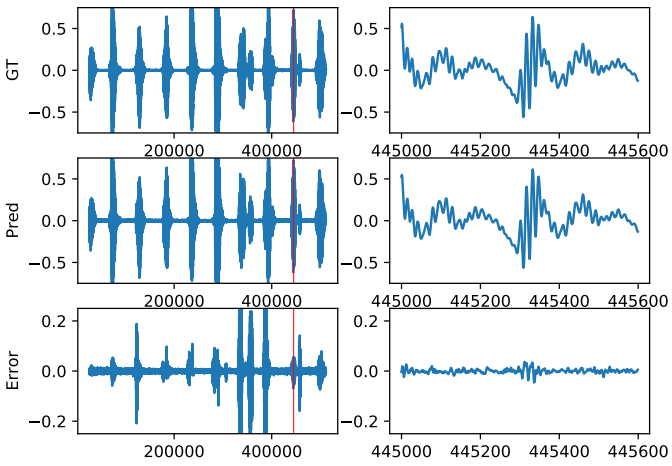


(b) SIREN+S.T.

Fig. 3. Regressed waveforms of the INRs trained with and without derivative supervision on the clip of *Bach*. From top to bottom: ground truth waveform, regressed waveform and error detected. We zoom in the waveform marked with red at the right column accordingly.



(a) SIREN [10]



(b) SIREN+S.T.

Fig. 4. Regressed waveforms of the INRs trained with and without derivative supervision on the clip of *Counting*. From top to bottom: ground truth waveform, regressed waveform and error detected. We zoom in the waveform marked with red at the right column accordingly.

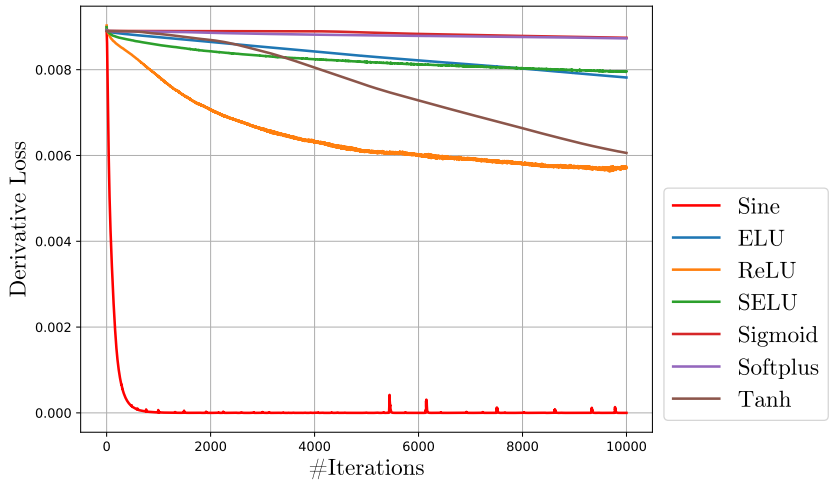


Fig. 5. Derivative loss of different activation functions.

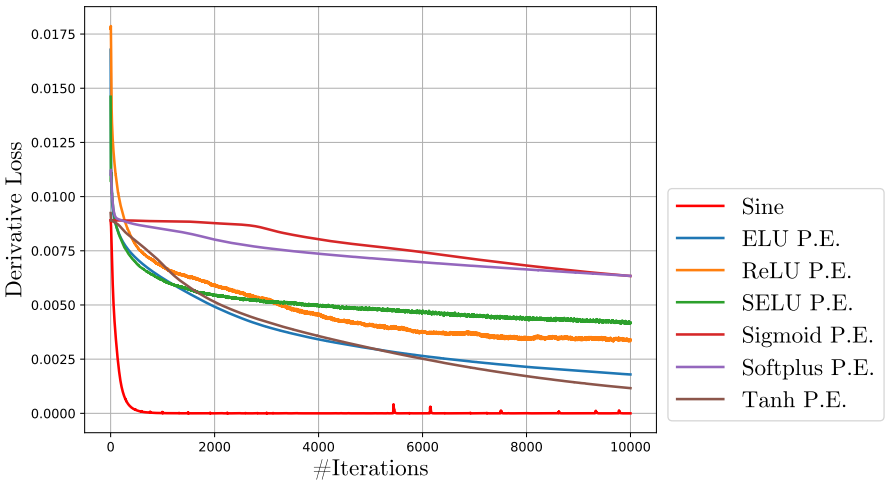


Fig. 6. Derivative loss of different activation functions with positional encoding. P.E. means positional encoding.

References

1. Agarap, A.F.: Deep learning using rectified linear units (relu). arXiv preprint arXiv:1803.08375 (2018) 4
2. Agustsson, E., Timofte, R.: Ntire 2017 challenge on single image super-resolution: Dataset and study. In: The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops (July 2017) 5
3. Bevilacqua, M., Roumy, A., Guillemot, C., Alberi-Morel, M.L.: Low-complexity single-image super-resolution based on nonnegative neighbor embedding (2012) 1, 3
4. Clevert, D.A., Unterthiner, T., Hochreiter, S.: Fast and accurate deep network learning by exponential linear units (elus). arXiv preprint arXiv:1511.07289 (2015) 4
5. Glorot, X., Bengio, Y.: Understanding the difficulty of training deep feedforward neural networks. In: Proceedings of the thirteenth international conference on artificial intelligence and statistics. pp. 249–256. JMLR Workshop and Conference Proceedings (2010) 4
6. He, K., Zhang, X., Ren, S., Sun, J.: Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In: Proceedings of the IEEE international conference on computer vision. pp. 1026–1034 (2015) 1, 4
7. Klambauer, G., Unterthiner, T., Mayr, A., Hochreiter, S.: Self-normalizing neural networks. Advances in neural information processing systems **30** (2017) 4
8. Mildenhall, B., Srinivasan, P.P., Ortiz-Cayon, R., Kalantari, N.K., Ramamoorthi, R., Ng, R., Kar, A.: Local light field fusion: Practical view synthesis with prescriptive sampling guidelines. ACM Transactions on Graphics (TOG) **38**(4), 1–14 (2019) 1, 5
9. Mildenhall, B., Srinivasan, P.P., Tancik, M., Barron, J.T., Ramamoorthi, R., Ng, R.: Nerf: Representing scenes as neural radiance fields for view synthesis. In: European conference on computer vision. pp. 405–421. Springer (2020) 1, 2, 5, 6
10. Sitzmann, V., Martel, J., Bergman, A., Lindell, D., Wetzstein, G.: Implicit neural representations with periodic activation functions. Advances in Neural Information Processing Systems **33**, 7462–7473 (2020) 1, 2, 3, 4, 6, 8, 9
11. Tancik, M., Srinivasan, P., Mildenhall, B., Fridovich-Keil, S., Raghavan, N., Singhal, U., Ramamoorthi, R., Barron, J., Ng, R.: Fourier features let networks learn high frequency functions in low dimensional domains. Advances in Neural Information Processing Systems **33**, 7537–7547 (2020) 3
12. Yen-Chen, L.: Nerf-pytorch. https://github.com/yenchenlin/nerf-pytorch/ (2020) 6
13. Yüce, G., Ortiz-Jiménez, G., Besbinar, B., Frossard, P.: A structured dictionary perspective on implicit neural representations. arXiv preprint arXiv:2112.01917 (2021) 3