Learned Initializations for Optimizing Coordinate-Based Neural Representations: Supplemental Material

1. Implementation details

We found that modifying the weight initialization for these coordinate-based networks drastically changed their convergence behavior during test-time optimization. As a result, we tuned the optimization method and hyperparameters for each part of each experiment (using held-out validation sets) in order to provide the fairest possible comparison and to not bias the results against the non-meta-learned initializations. For example, we often found that SGD outperformed Adam when doing test-time optimization using meta-learned initializations, but that Adam was significantly better than SGD with a standard random initialization.

All experiments are implemented in JAX [1]. Each experiment is trained on either a single NVIDIA V100, 2080 Ti, or 3080 Ti. In all cases where the Adam optimizer [5] is used, we keep the standard parameter choices for $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 10^{-8}$.

1.1. Image regression

For this task we use a SIREN [8] architecture ($\omega_0 = 200$) with 5 layers of 256 channels each. For the randomly initialized *Standard* baseline, we use the specific initialization procedure as proposed in the SIREN paper.

MAML [3] is trained for 150K iterations. Each iteration has an outer batch size of 3 target images. The inner batch contains all pixels of the target image. The outer loop uses the Adam optimizer with learning rate of 10^{-5}. The inner loop performs two steps of gradient descent with a learning rate of 10^{-2}.

We additionally meta-learn another initialization using Reptile [7]. We use the same learning rates as in MAML but with an outer batch size of 10 target images. We report the Reptile reconstruction accuracy in Table 1. We note that Reptile also outperforms the non-meta-learned weights.

During test-time optimization, we use gradient descent with learning rate of 10^{-2} when starting from the MAML initial weights. For the baseline methods (*Standard*, *Mean*, *Matched*, *Shuffled*) we used Adam with learning rate of 10^{-4}, which performed significantly better than than gradient descent.

Init. Method	2 Step PSNR ↑	# of iters to match ↓
Standard	10.88	37.92 ± 6.31
Mean	14.48	25.59 ± 4.57
Matched	13.73	26.32 ± 4.17
Shuffled	16.29	25.80 ± 4.02
Reptile	25.55	$\mathbf{9.86 \pm 7.42}$
MAML	**30.37**	-

Table 1. Image reconstruction results with meta-learning results for both MAML and Reptile. MAML performs best, but Reptile also outperforms the non-meta-learned weights.

1.2. CT reconstruction

For this task we use an MLP with 5 layers of 256 channels each. The network uses a ReLU activation after each layer with the exception of the last layer, which has a sigmoid activation. Prior to inputting the coordinates into the network, we encode them using random Fourier features sampled from a normal distribution with $\sigma = 30$, as was done in Tancik *et al*. [9].

Reptile [7] is trained for 100K iterations. Each iteration has an outer batch size of 1. The inner batch contains 20 CT projections, each with 256 measurements, taken from a randomly sampled direction. The outer loop uses the Adam optimizer with learning rate of 5×10^{-5}. The inner loop performs 12 inner loop steps of gradient descent with a learning rate of 10^1.

We perform test-time optimization experiments with different numbers of supervision views to compare reconstruction quality. We found that the models are more prone to overfitting when fewer views are provided. We tune the learning rate and number of gradient steps for each initialization method according to a held-out set of 16 validation images. We report all of the test-time optimization hyperparameters in Table 2.

1.3. ShapeNet [2] view synthesis

We use a simplified NeRF [6] model for our view synthesis tasks. This model uses a single network rather than two networks (coarse and fine), and we do not provide view directions as input. The network is an MLP with 6 lay-

ers, each with 256 channels and ReLU activations. As in NeRF [6], we apply a positional encoding to each input coordinate with the form

$$\bigcup_{i=0}^{N} \left\{ \cos\left(2^{fi/N} x\right), \sin\left(2^{fi/N} x\right) \right\}, \quad (1)$$

with $N = 20$ encodings and log-max frequency $f = 8$. We accumulate 128 samples per ray for rendering.

Reptile is trained for 100K iterations with an outer batch size of 1. The inner loop step optimizes over a batch of 128 rays. We perform 32 inner loop steps for every outer loop step. The outer loop uses the Adam optimizer with learning rate 5×10^{-4} for the *Chairs* scenes and 5×10^{-5} for the *Lamps* and *Cars* scenes.

The test-time optimization parameters vary depending on the scene and the number of views available during meta-learning. Each experiment uses an inner batch of 64 rays. The *Shuffled* and *Matched* initializations are computed based on the *MV Meta* weights. For the 25 view chair reconstruction, we use stochastic gradient descent with a learning rate of 10^{-1} for the Reptile initialization; for the standard initialization, we use Adam with a learning rate of 10^{-4}. The test-time hyper parameters for the single view experiments are listed in Table 3.

1.4. Phototourism [4] view synthesis

We use the same architecture as described in §1.3. Reptile is trained for 150K iterations with an outer batch size of 1. The inner loop step optimizes over a batch of 64 rays, with 128 volume rendering samples per ray. The outer loop uses the Adam optimizer with a learning rate of 5^{-4}. We train with 64 inner loop steps using gradient descent with a learning rate of 10. We compare to *Basic NeRF* which has the same setup, but only one inner step. For *Basic NeRF* we train *Trevi* for 60K iterations, *Brandenburg* for 100K iterations, and *Sacre Coeur* for 200K iterations. To transfer the appearance of a new photo during test-time optimization, we take 150 gradient steps with a learning rate of 10.

2. Weight space interpolation

We find that linearly interpolating between networks in weight space produces meaningful outputs when using meta-learned weights. Figure 1 shows interpolation between networks trained to represent images, and Figure 2 shows interpolations between networks that are trained to reconstruct a Phototourism landmark.

3. Acknowledgements

MT is funded by an NSF fellowship and a Berkeley DeepDrive grant. BM is funded by Google through the BAIR Commons Program. Google University Relations provided a generous donation of GCP compute credits. We thank Ruichao Ren from NVIDIA for donating GPU hardware.

References

[1] James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, and Skye Wanderman-Milne. JAX: composable transformations of Python+NumPy programs, 2018.

[2] Angel X. Chang, Thomas Funkhouser, Leonidas Guibas, Pat Hanrahan, Qixing Huang, Zimo Li, Silvio Savarese, Manolis Savva, Shuran Song, Hao Su, Jianxiong Xiao, Li Yi, and Fisher Yu. Shapenet: An information-rich 3d model repository. Technical report, 2015.

[3] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. *ICML*, 2017.

[4] Yuhe Jin, Dmytro Mishkin, Anastasiia Mishchuk, Jiri Matas, Pascal Fua, Kwang Moo Yi, and Eduard Trulls. Image matching across wide baselines: From paper to practice. *International Journal of Computer Vision*, pages 1–31, 2020.

[5] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *ICLR*, 2015.

[6] 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. *ECCV*, 2020.

[7] Alex Nichol, Joshua Achiam, and John Schulman. On first-order meta-learning algorithms. *arXiv preprint arXiv:1803.02999*, 2018.

[8] Vincent Sitzmann, Julien N.P. Martel, Alexander W. Bergman, David B. Lindell, and Gordon Wetzstein. Implicit neural representations with periodic activation functions. *NeurIPS*, 2020.

[9] Matthew Tancik, Pratul P. Srinivasan, Ben Mildenhall, Sara Fridovich-Keil, Nithin Raghavan, Utkarsh Singhal, Ravi Ramamoorthi, Jonathan T. Barron, and Ren Ng. Fourier features let networks learn high frequency functions in low dimensional domains. *NeurIPS*, 2020.

	Opt. method		Number of steps			
	Adam	LR	1 View	2 Views	4 Views	8 Views
Standard	✓	5×10^{-4}	50	100	250	1000
Mean	✓	10^{-5}	25	50	100	1000
Matched	✓	5×10^{-4}	50	100	500	1000
Shuffled	✓	5×10^{-4}	50	100	250	1000
Meta	✗	10^{1}	50	100	1000	1000

Table 2. Hyper-parameters for CT test-time optimization. Each value is tuned on a held-out validation set.

	Chairs			Cars			Lamps		
	Adam	LR	Steps	Adam	LR	Steps	Adam	LR	Steps
Standard	✓	10^{-5}	1000	✓	5×10^{-5}	2000	✓	5×10^{-5}	2000
Matched	✓	10^{-4}	2000	✓	5×10^{-5}	2000	✓	5×10^{-5}	2000
Shuffled	✓	10^{-4}	2000	✓	5×10^{-5}	2000	✓	5×10^{-5}	2000
MV Meta	✗	10^{-1}	1000	✗	5×10^{-1}	2000	✗	5×10^{-1}	2000
SV Meta	✗	5×10^{-1}	1000	✗	5×10^{-1}	2000	✗	5×10^{-1}	2000

Table 3. Hyper-parameters for ShapeNet test-time optimization from a single view. Each value is tuned on a held-out validation set.

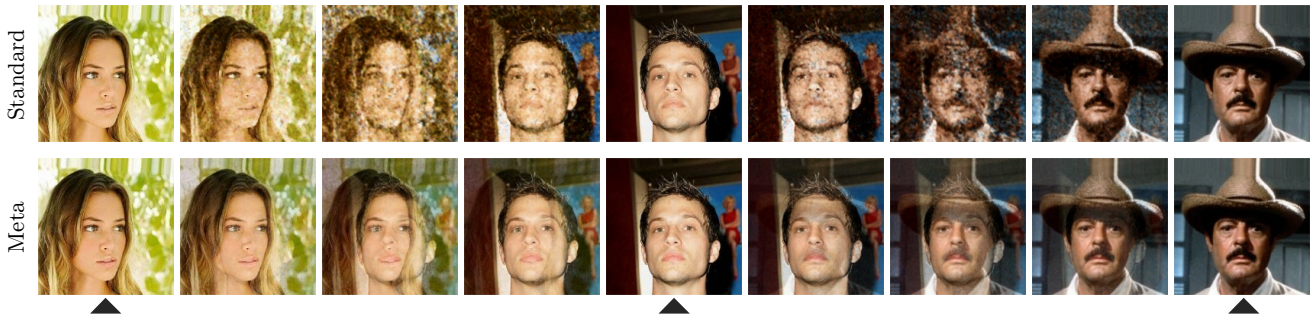

Figure 1. Weight space interpolation for networks optimized to represent 2D images. We use test-time optimization to fit network weights for three different images (denoted with arrows), then linearly interpolate between those weight values and visualize the resulting outputs. When test-time optimization begins from a standard random initialization (*Standard*, top), weight space interpolation produces displeasing artifacts, but when it begins from a meta-learned initialization (*Meta*, bottom) the resulting outputs maintain an image-like appearance.

Figure 2. Appearance interpolation on the Trevi Fountain scene from the Phototourism dataset [4]. We render the scene from a single fixed camera pose. Each corner of the grid represents a NeRF network that has been test-time optimized to match the appearance of a single image (starting from the meta-learned initial weights θ_0^*). We linearly interpolate between these networks in weight space to render the grid of images shown here. The image in the center is produced by directly rendering the meta-learned initial weights (with no test-time optimziation), representing an "average" appearance for the scene. Please see the supplemental video for an animated version of this figure with a moving camera path.