## Appendix

## A Use of MetaSin Beyond Image Prediction

In addition to our main results presented in Section 4 of the paper, we also performed various exploratory experiments to investigate further application cases of the MetaSin activations. The experiments cover image classification where we show favorable results of using convolutional MetaSin networks over baseline ReLU networks, as well as various overfitting experiments to explore the use of MetaSin activations with MLPs. We present our preliminary findings here with the hope of motivating further investigation in these directions.

In all MLP experiments, we use MetaSin with $K = 10$ sine components, and distribute the frequencies evenly across the range $[1, 35]$. The initialization of the remaining parameters follows the description provided in Section 3.

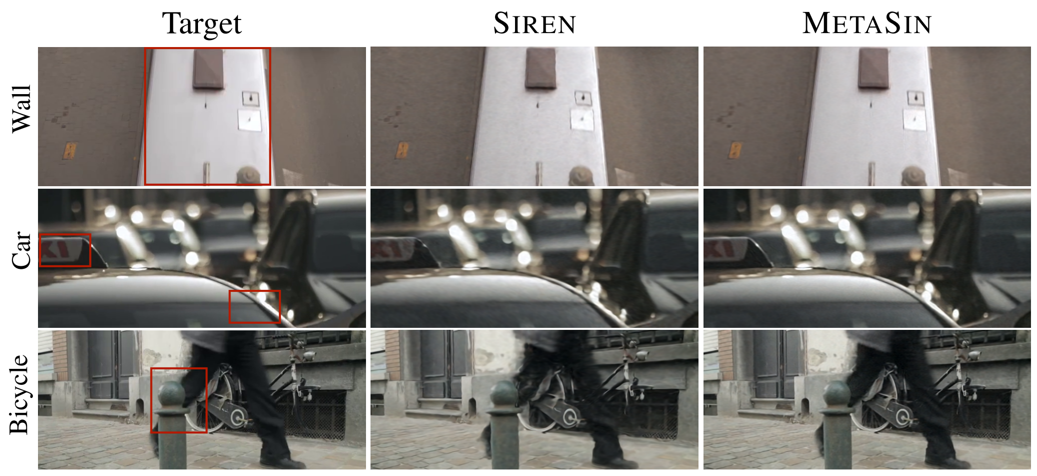


Figure 5: Visualization of selected reconstructed frames of the video. To fully appreciate the details and visual cues presented in the figure, we recommend visualizing the figures in color and zooming in for a more comprehensive analysis.

| Video Dataset | Activation | Model | Channels | Depth | MSE ↓ | PSNR (dB) ↑ |
|---|---|---|---|---|---|---|
| Cat | Siren | MLP | 1024 | 3 | $2.60 \cdot 10^{-3}$ | 32.26 |
| | MetaSin | MLP | 1024 | 3 | $\mathbf{1.96 \cdot 10^{-3}}$ | **33.32** |
| Bikes | Siren | MLP | 1024 | 3 | $1.69 \cdot 10^{-3}$ | 34.50 |
| | MetaSin | MLP | 1024 | 3 | $\mathbf{5.58 \cdot 10^{-4}}$ | **39.32** |

Table 7: Results on implicit video representation.

### A.1 Video Overfitting using MetaSin MLP

In this section, we present the overfitting experiments on videos using an MLP network with 3 hidden layers and 1024 neurons to overfit a video. The input is a 3 dimensional vector consisting of a frame ID and pixel coordinate, and the output is the RGB value of the corresponding pixel. The MetaSin network improves the PNSR by 1-4 dB compared to the Siren baseline as shown in Table 7.

Overall the *Cat* dataset has more fine details, making it more challenging to accurately represent these intricate details. Consequently, the PSNR for the *Cat* dataset is comparatively lower in comparison to the *Bikes* dataset. The results are depicted in Figure 5. While ReLU networks struggle to represent the signal, both Siren network and MetaSin can reconstruct the frames. Notably, the MetaSin network reconstructs more details, see Figure 5 (Bicycle) and has less noise, as we see in Figure 5 (Wall and Car). It is worth mentioning that various techniques, such as positional encoding or Fourier

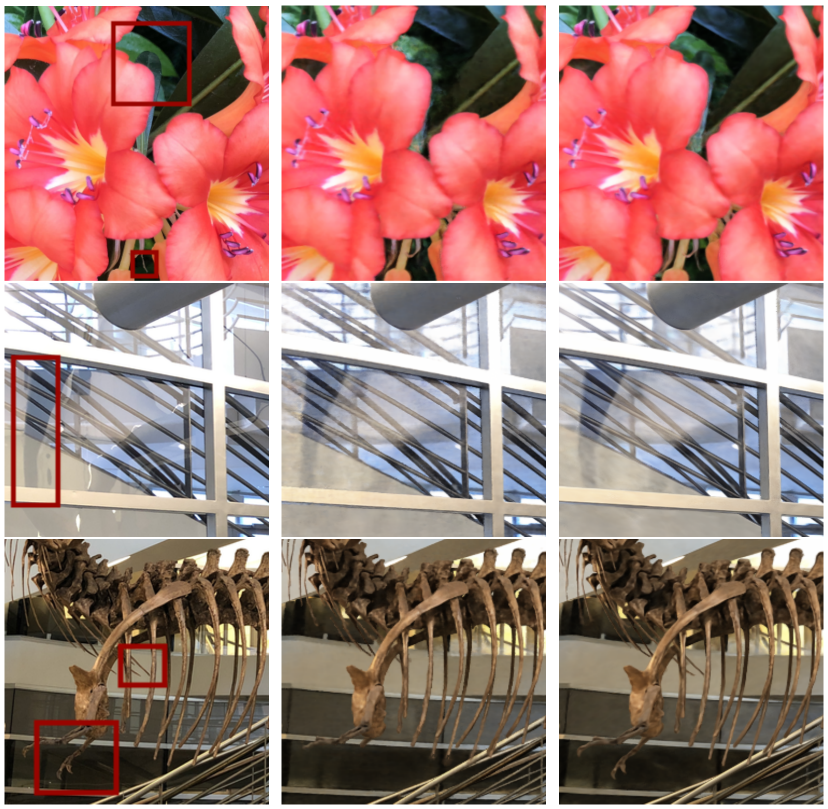


Figure 6: Zoomed qualitative results on static forward scenes [28]. Comparison between ground-truth, NeRF and METASIN-10 model.

features [43, 42], as well as utilizing latent codes [31], have shown significant improvements in RELU networks. However, exploring the impact of these techniques when applied in conjunction with METASIN activation is a direction we defer to future work.

| Method | Leaves | Orchids | Flower | T-Rex | Horns |
|---|---|---|---|---|---|
| NeRF [29] | 20.92 | 20.36 | 27.40 | 26.80 | 27.45 |
| METASIN-10 | **21.46** | **20.52** | **28.20** | **26.81** | **27.98** |

Table 8: Per-scene PSNR↑ comparison. In this table, we report the results obtained by running NeRF PyTorch implementation, based on [46] and previously used in [21, 6], and the results of our proposed METASIN. Our method achieves higher PSNR in all tested scenes.

### A.2 Novel View Synthesis using METASIN MLP

*Neural Radiance Fields* [29] (NeRF) enable synthesizing novel views with only a sparse set of input views. NeRF methods exploit *positional encoding* to map the input into high-frequency space, which is the key to high-quality scene rendering, which we drop in our experiments and replace the standard RELU with a 10 components METASIN-10 activation. Frequencies and phases are initialized uniformly over $[1, 35]$ and $[0, 2\pi]$. The experiment setting is the same as [29], except that we extend the training process to 250K iterations instead of 200K, in order to make sure that METASIN models converge well. Our implementation is based on [46], The results are reported in Table 8. We also provide a comparison of qualitative results in Figure 6.

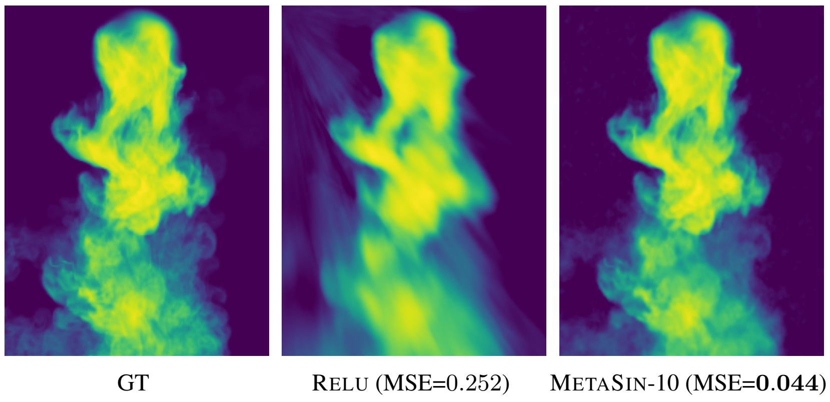


Figure 7: False-color visualization of the reconstructed density grid where METASIN is configured with $K = 10$

### A.3 Density Overfitting using METASIN MLP

In this section we present an overfitting experiment on a 3D voxel grid. The smoke scene we used in this experiment contains $175 \times 232 \times 175$ density values within range $[0, 1]$. Following the MLP architecture used in [39], we map the 3D coordinates to the corresponding density value using RELU and METASIN activated networks. Comparisons between the METASIN activated network and the baseline RELU model are presented in Figure 7.

## B Details of Overfitting Experiment with Convolutional Network

Table 9 shows further architectural details of the overfitting experiment we discuss in Section 3 of the main paper. We perform experiments with two base convolutional architectures, a shallow and a deep network. M-SIREN refers to the amplitude modulated SIREN [27], which has significantly more parameters due to the introduction of an additional amplitude modulation network.

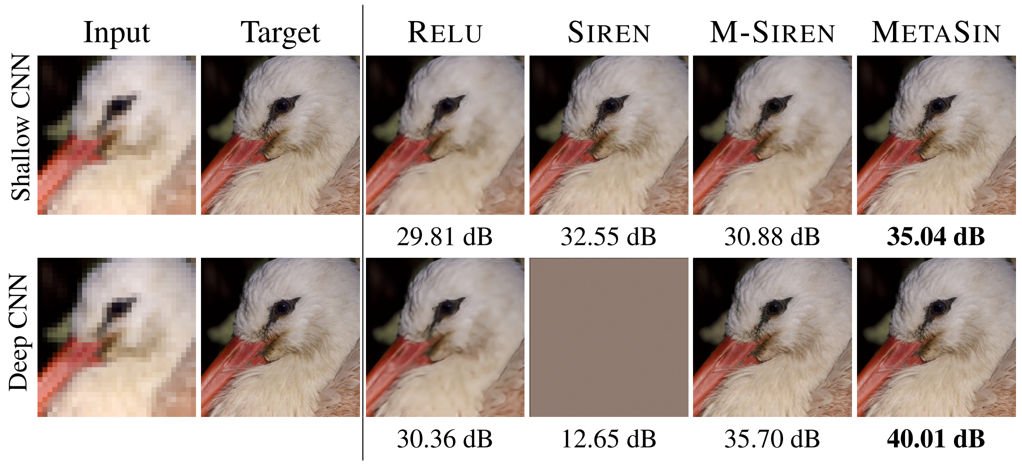


Figure 8: Peak Signal-to-Noise Ratio (PSNR) comparison of different activations employed on CNNs for image up-sampling (with 16x of upsampling factor).

Figure 8 shows a visual comparison of reconstructed images by the different versions of both base networks. Both the shallow and deep versions of the METASIN network can be trained reliably (unlike their sin activated counterparts that diverge in the case of the deep architecture). The METASIN networks come with relatively little parameter overhead (the relative overhead becomes even smaller with larger networks that have more channels, as discussed in Appendix E), and produce visually sharper results that also are closer to the reference in terms of PSNR.

| CNN Size | Activation | # Param. | Depth | PSNR (dB) ↑ |
|---|---|---|---|---|
| Shallow | RELU | 75 K (1x) | 4 | 29.81 |
| | SIREN [39] | 76 K (1x) | 4 | 32.55 |
| | M-SIREN [27] | 281 K (3.7x) | 4 | 30.88 |
| | METASIN (ours) | 82 K (1.1x) | 4 | **35.04** |
| Deep | RELU | 408 K (1x) | 13 | 30.36 |
| | SIREN [39] | 408 K (1x) | 13 | 12.65 |
| | M-SIREN [27] | 651 K (1.6x) | 13 | 35.70 |
| | METASIN (ours) | 432 K (1.1x) | 13 | **40.01** |

Table 9: Comparison of different activations employed on CNNs trained for image up-sampling (16x).

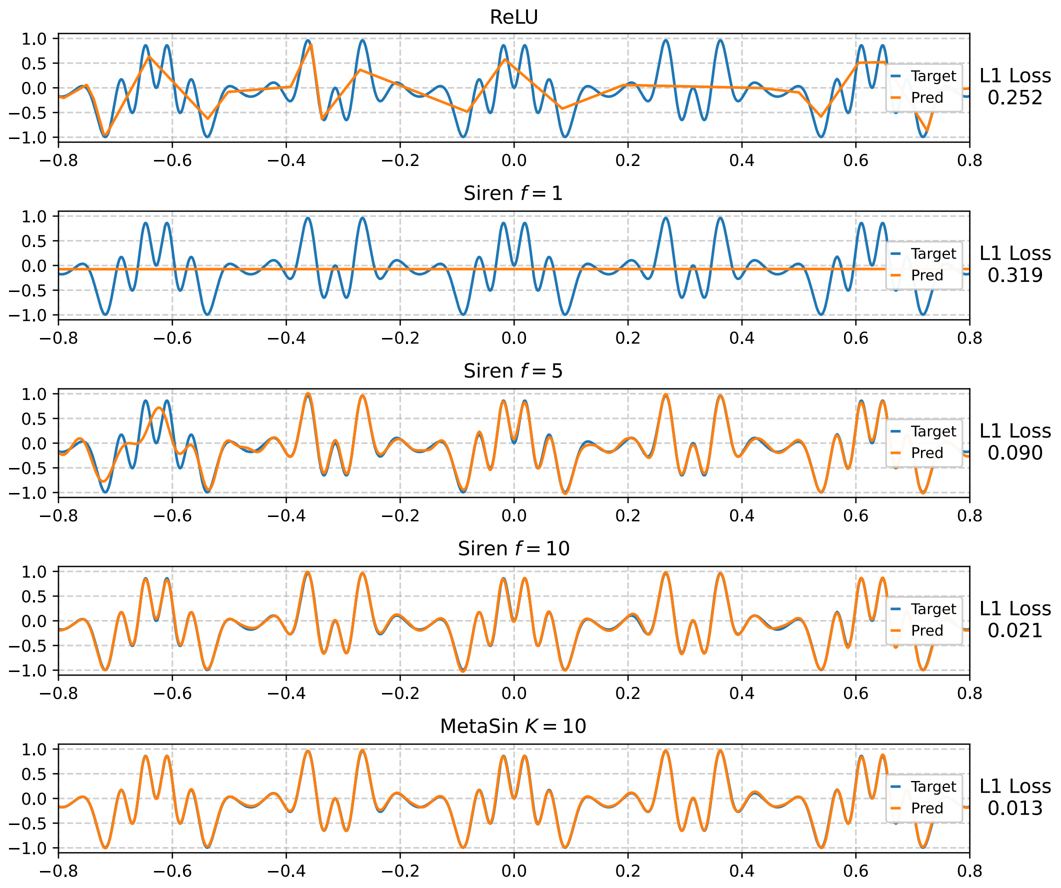


Figure 9: Representing $y(t) = \sin(10t + 5\sin(10t)) \times \sin(10\sin(10t))$ with 3 layer FC networks

## C Sensitivity to Frequency Initialization

Although the $\sin$ activation function has promising properties as outlined in the main paper, it is known that they can be extremely difficult to train [39, 49].

We illustrate this in a toy experiment where we try to reconstruct the signal $y(t) = \sin(10t + 5\sin(10t)) \times \sin(10\sin(10t))$ when the input is $x(t) = t$. Most of the energy of $y(t)$ is distributed between frequencies between 10 and 170. We sample $t$ from the normal distribution and train fully connected networks with 1 input neuron, 3 layers having 20 neurons each, and 1 output neuron. In one of the networks, we use RELU activation in between fully connected layers except for the last one. In the remaining two networks, we use $\sin(fx)$. In one we initialize $f = 1$, in the other one we initialize $f = 5$, and $f = 10$. We use $L1$ loss and Adam as the optimizer. The results are shown in Figure 9. As it is seen, RELU-network can reconstruct part of the signal. The sin-network with the initialization $f = 1$ does not learn at all, while the network initialized with $f = 5$ is able to match $y(t)$, and with $f = 10$ the model fits the original signal reasonably well. This suggests the importance of initialization and the difficulty of training of the sin activation function.

On the other hand, a METASIN network with the frequency parameters from Equation 4 initialized as $f_j^{[l]} = j$ and $K = 10$ effectively covers a wide range of frequencies including the one that is important to represent the signal. Consequently, using a METASIN we are able to lower the loss to $0.013$ and eliminate the trial and error process for initializing frequencies.

In Figure 10 we additionally plot the results of an experiment where we sweep through a wide range of $w_0$ values for a fully-connected SIREN network that was trained to overfit to a single image. The plot illustrates that only a narrow range among possible initial $w_0$ values results in a plausible training error, confirming the importance of initialization in SIREN networks.

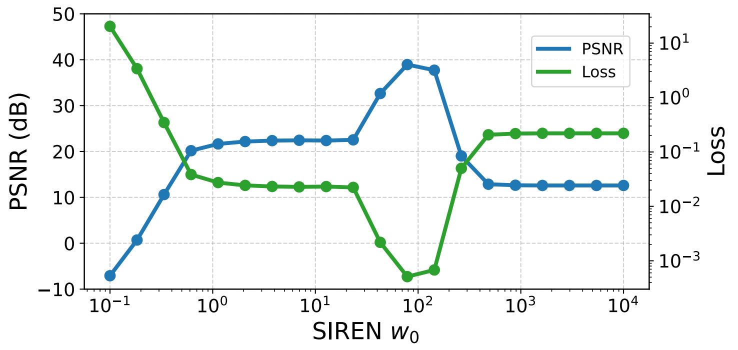


Figure 10: Sensitivity of SIREN w.r.t. base frequency $w_0$. In this figure, we reported the PSNR and MSE Loss values across a wide range of SIREN's base frequency $w_0$ values. For each $w_0$, we trained a SIREN network and showed that only a restricted set of frequencies (around $10^2$) converges to sufficiently good local minima. Specifically, in this experiment, we train the networks to up-sample the image reported in Figure 8, with an up-scale of $\times 16$ for $100,000$ iterations using the Adam optimizer with a learning rate of $10^{-5}$.

## D Connection to Fourier Series

Well-behaved functions, such as continuous functions can be expressed as a Fourier series with finitely many terms. Let us define a function $\phi$ as a Fourier series:

$$\phi(\mathbf{x}) = a_0 + \sum_{j=1}^{K} \left( a_j \cos\left(\frac{2\pi j}{P}\mathbf{x}\right) + b_j \sin\left(\frac{2\pi j}{P}\mathbf{x}\right)\right), \tag{5}$$

where $K \in \mathbb{N}^+$ is finite, and $P$ denotes length of one period. Using the harmonic addition rule, i.e.

$$\begin{aligned} &a\cos\alpha + b\sin\alpha = c\cos(\alpha + r), \text{ where} \\ &c = \operatorname{sgn}(a)\sqrt{a^2+b^2} \\ &r = \arctan\left(-\frac{b}{a}\right), \end{aligned} \tag{6}$$

we can rewrite Equation 5 as:

$$\phi(\mathbf{x}) = a_0 + \sum_{j=1}^{K} c_n \cos\left(\frac{2\pi j}{P}\mathbf{x} + r\right). \tag{7}$$

Substituting in the identity $\cos\alpha = \sin\left(\alpha + \frac{\pi}{2}\right)$ we get:

$$\phi(\mathbf{x}) = a_0 + \sum_{j=1}^{K} c_n \sin\left(\frac{2\pi j}{P}\mathbf{x} + \frac{\pi + 2r}{2}\right). \tag{8}$$

Dropping the constant term we obtain the following:

$$\begin{aligned} \phi(\mathbf{x}) &= \sum_{j=1}^{K} c_j \sin\left(f_j \mathbf{x} + p_j\right), \text{ where} \\ f_j &= \frac{2\pi j}{P}, \text{ and} \\ p_j &= \arctan\left(-\frac{b_j}{a_j}\right) + \frac{\pi}{2}. \end{aligned} \tag{9}$$

From Equation 9 we can see that any Fourier Series can be expressed as a MetaSin activation

$$\phi(\mathbf{x}) = c_0 \mathrm{ReLU}(\mathbf{x}) + \sum_{j=1}^{K} c_j \sin\left(f_j \mathbf{x} + p_j\right), \tag{10}$$

by setting $c_0 = 0$ and the rest of its parameters $\{c_j, f_j, p_j; j \in [1, K]\}$ accordingly.

## E MetaSin Overhead

MetaSin parameter overhead formula for a convolutional layer compared to ReLU baseline is as follows:

$$\text{overhead} = \frac{3 \times K + 1}{\text{kernel-size}^2 \times \text{input-channels}}. \tag{11}$$

For example if we have a layer with kernel-size $= 3$ with 128 input-channels, then overhead for $K = 10$ is 0.026. Which means we have approximately $2.6\%$ more parameters. If input-channels is 32 the overhead becomes $10.08\%$. On the other hand for kernel-size $= 5$ and 512 input-channels the overhead is $0.24\%$.

## F Details on C++/CUDA Implementation

To account for the inefficiency of a native Python API implementation of MetaSin we implemented custom-optimized fused CUDA kernels for both forward and backward functions of the MetaSin activation in C++. Our implementation can be integrated into the PyTorch and TensorFlow Python APIs. Throughout the development process we also tested native automatic compilation functionalities provided by both frameworks (specifically: jit, torch.compile for PyTorch, and jit and XLA in TensorFlow). While notable improvements over the baseline Python API implementation can be made through the use of these out-of-the-box facilities, we obtained the best performances in terms of speed and memory consumption using our custom-designed CUDA functions.

Some of the techniques we utilized in our code are as follows: in order to optimize the memory footprint and the inference speed of the MetaSin, we remove the intermediate quantities that the autograd engine computes and instead compute the output and gradient tensors directly from the input and the MetaSin parameters. Moreover, we further optimized the computation speed with improved caching and reduction strategy on warp and block levels. Meanwhile, we have exploited a 2-level reduction strategy based on *pairwise summation* in our backward kernel. In this way, we could avoid numerical errors in gradient tensors and achieve comparable accuracy to autograd in float32 precision.

The reduction in computational overhead using the aforementioned optimizations enabled running the compute-intensive experiments we present throughout the paper in a feasible manner, and we believe demonstrates the viability of MetaSin activations for most practical tasks. That being said, other optimizations that we were not able to explore due to time constraints could help reduce the overhead even further, which is an interesting direction for future research.

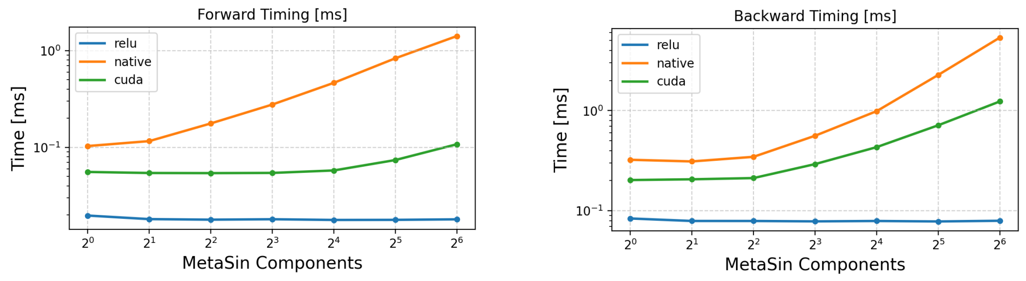


Figure 11: Effect of METASIN components to latency.

We also investigated the computational scaling behavior of our implementation with respect to the number of $sin$ components $K$. To this end, we ran a benchmark experiment, in which we compared forward and backward latencies of RELU, METASIN Native, and METASIN CUDA functions on the same input tensors using different $K$ values. The results can be found in Figure 11.

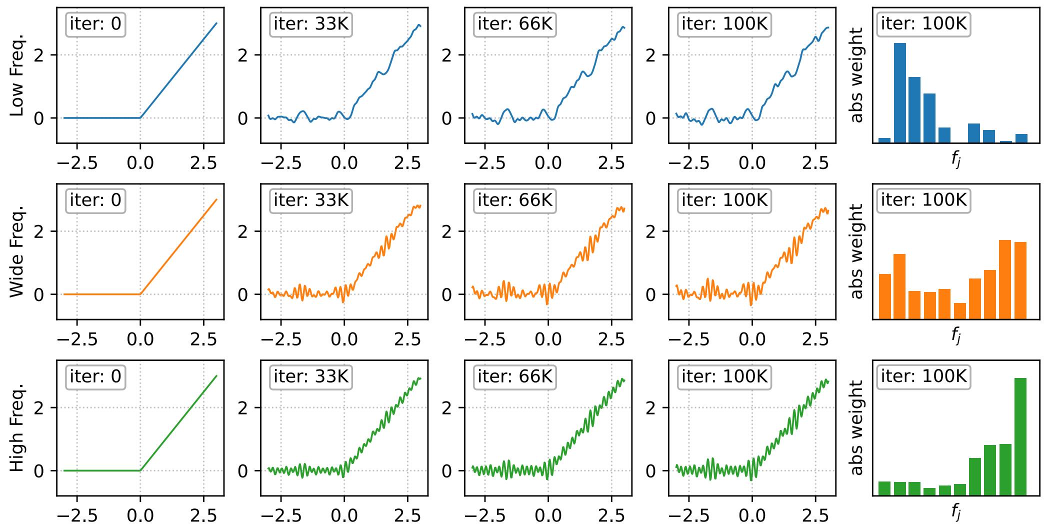


Figure 12: METASIN activation shape evolution over time and final distribution of frequency parameters. In this figure, we report the behavior of three activated neurons (of the same layer) over the training process.

## G Evolution of METASIN Activations during Training

Figure 12 shows illustrative examples of the evolution of METASIN activations during the course of training in a toy example. In this experiment the weight of the RELU component is set to 1, whereas the amplitudes of all $sin$ components are set to 0. Thus a METASIN activation starts out as RELU, but then evolves into various shapes depending on the training of the shape parameters via backpropagation. It is worth noting that the shape of a METASIN activation tends to not change after the initial phase of training, which agrees with the observation that weights change very slowly in overparameterized networks. Moreover, the different rows in the figure show three different METASIN neurons of the same layer that interestingly converge to a complementary frequency response that can capture different frequency components of the input signal.

To shed some light on the shapes that METASIN activations take in a more practically relevant setting with and without KD-Bootstrapping, we additionally produced a visualization of the METASIN shapes from the resampling network described in Section 4.1. The METASIN shapes are obtained from the first and last blocks of models trained from scratch and with KD Bootstrapping. This visualization in Figure 13 shows that, while there is significant local variation between individual METASIN activations, globally the rough RELU shape is still discernible even without any involvement from the RELU teacher.

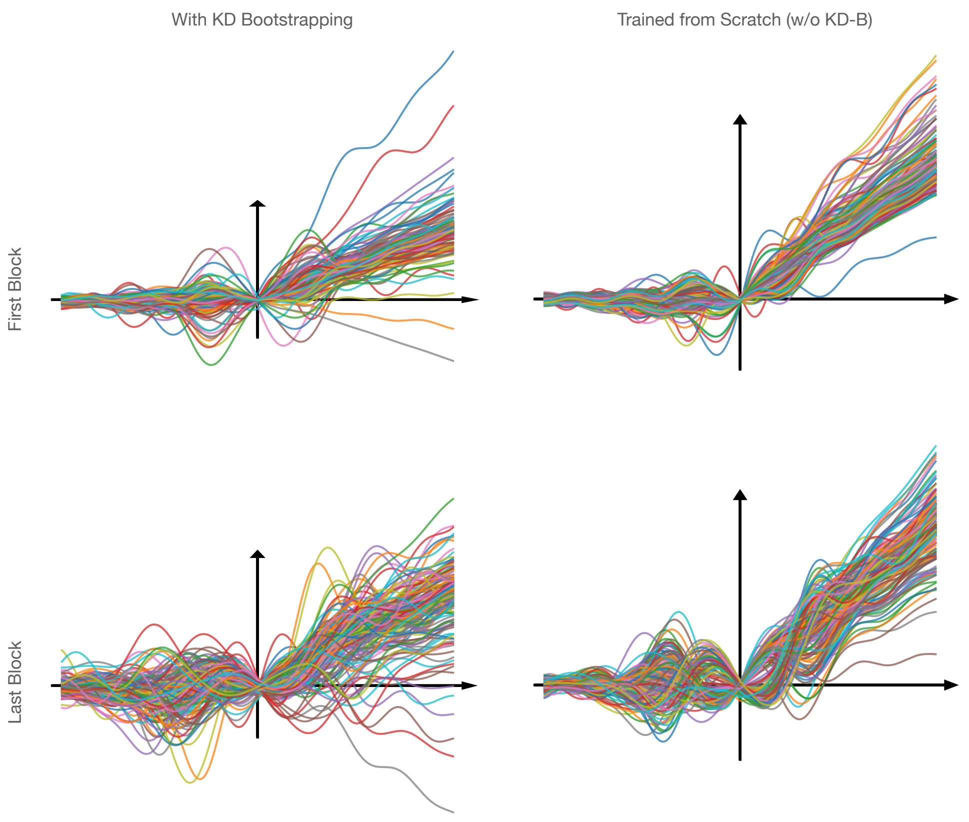


Figure 13: Visualization of METASIN shapes with and without KD bootstrapping.

## H Comparison with Ensemble Activations

We ran additional experiments using ensemble activations as described in a recent survey [3], namely Adaptive Blending Units (abu), Variable AFs (vaf), Adaptive AFs (aaf) in the same experimental setting that we used to generate the first row of Figure 1 in the main paper. We present the best results we obtained from multiple runs with different initializations in Table 10, also including RELU and METASIN as references:

| | abu | vaf | aaf | RELU | METASIN |
|---|---|---|---|---|---|
| PSNR[db] | 29.79 | 28.98 | 29.06 | 29.81 | **35.04** |

Table 10: Comparison of various enseble activations in an image overfitting setting.

We also ran two experiments in image resampling setting using abu and aaf versions of the model from Section 4.1, which we present in Table 11:

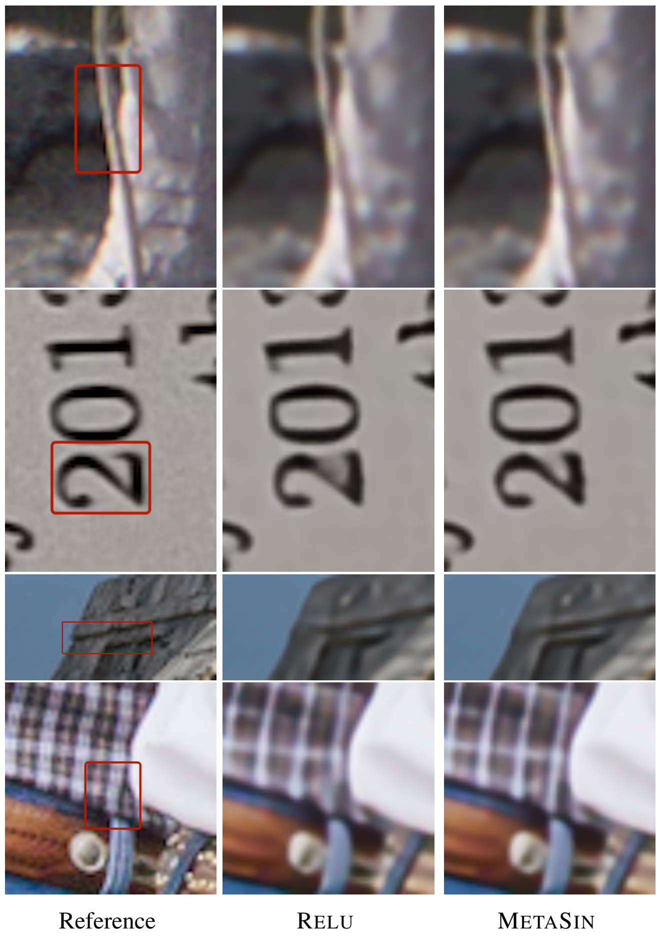


Figure 14: Additional upsampling results comparing the state-of-the-art model with RELU activations [5] and the corresponding METASIN network.

| | Activation | | | |
|---|---|---|---|---|
| Factor | abu | aaf | RELU | METASIN w/ KD-B |
| $P_{\times 2}$ | 33.02 | 32.97 | 33.03 | **33.26** |
| $P_{\times 3}$ | 29.31 | 29.28 | 29.36 | **29.58** |
| $P_{\times 4}$ | 27.09 | 27.10 | 27.09 | **27.29** |

Table 11: Comparison of various ensemble activations for image resampling

## I Effect of Setting $K$ to a high number

We note that further improvements in terms of model accuracy can be made by pushing $K$ beyond what we present in the paper. We did not go beyond 10-12 as throughout our experiments we observed that doing so often resulted in diminishing returns. As a more concrete example we provide results from a resampling experiment in Table 12 (Note that these results are from a preliminary run with slightly inferior hyperparameters compared to the corresponding experiments we report in the main paper, and the training was stopped prematurely after 900K iterations instead of the full 2M - hence the lower PSNRs compared to Table 2). These results suggest that choosing $K > 10$ would indeed make sense in cases where the absolute best quality is desired and additional latencies are tolerable, but overall $K \approx 10$ seems to be the happy medium.

| Activation/Upsample Factor | $P_{\times 2}$ | $P_{\times 3}$ | $P_{\times 4}$ |
|---|---|---|---|
| METASIN-10 | 33.09 | 29.42 | 27.17 |
| METASIN-20 | 33.14 | 29.45 | 27.20 |

Table 12: The effect of setting $K$ to 10 and 20 on resampling quality

## J Classification Experiment with METASIN Model Trained from Scratch

To investigate the effect of KD-Bootstrapping in the classification setting, we trained various Wide-ResNets with original RELU activations and with METASIN activations entirely from scratch and without using KD-Bootstrapping. Table 13 presents test accuracy on CIFAR-100.

| Model/Activation | WRN-16-2 | WRN-28-2 | WRN-40-2 | WRN-40-4 |
|---|---|---|---|---|
| RELU | 72.85 | 74.82 | 75.95 | 78.99 |
| METASIN w/o KD-B | 71.82 | 73.88 | 75.44 | 78.44 |

Table 13: Comparison of METASIN and RELU models in image classification with training without KD-Bootstrapping. In this case METASIN models only become advantageous when train using KD-Bootstrapping, underlining the importance of the proposed training methodology.

In accordance with the above discussion these results underline the role of KD Bootstrapping for achieving the best results with METASIN networks. To give a concrete example: as we show above, when training from scratch WRN-16-2-METASIN at 71.82 accuracy is behind its RELU counterpart (WRN-16-2-ReLU) with accuracy 72.85. On the other hand, the last column of Table 6 shows that by distilling from a WRN-40-2 teacher, the accuracy of WRN-16-2-METASIN can be brought up to 74.10, whereas WRN-16-2-RELU achieves 73.65 accuracy using the same procedure.

## K Resampler Architecture and Additional Results

Our network architecture is based on the design proposed in [5]. We utilize a ProSR feature extractor [45], consisting of 3 residual blocks with 4 dense blocks within each residual block and a growth rate of 40 and 160 channels. The subsequent prediction layer is implemented as an MLP with 4 hidden layers and 256 neurons. In our experiments, this prediction layer remains unchanged. We apply METASIN activations only to the convolutional feature extractor. We present additional results in Figure 14 for the resampler experiment presented in Section 4.1.

## L DPCN Denoiser Architecture and Additional Results

The design of the U-Net architecture we use in Section 4.2 is detailed in Table 14. We also present

| Scale | Encoder ResBlocks | Encoder Channels | Decoder ResBlocks | Decoder Channels |
|---|---|---|---|---|
| 0 | 2 | 64 | 2 | 64 |
| 1 | 2 | 128 | 2 | 128 |
| 2 | 2 | 256 | 2 | 256 |
| 3 | 2 | 256 | 2 | 256 |
| 4 | 2 | 256 | 2 | 256 |

Table 14: The Configuration of the Standard DPCN U-Net. Scale 0 is the full-resolution scale. From one scale to the next one, the image resolution is halved in both dimensions.

additional results in Figure 15 to the results we presented in Section 4.2 in the main paper.

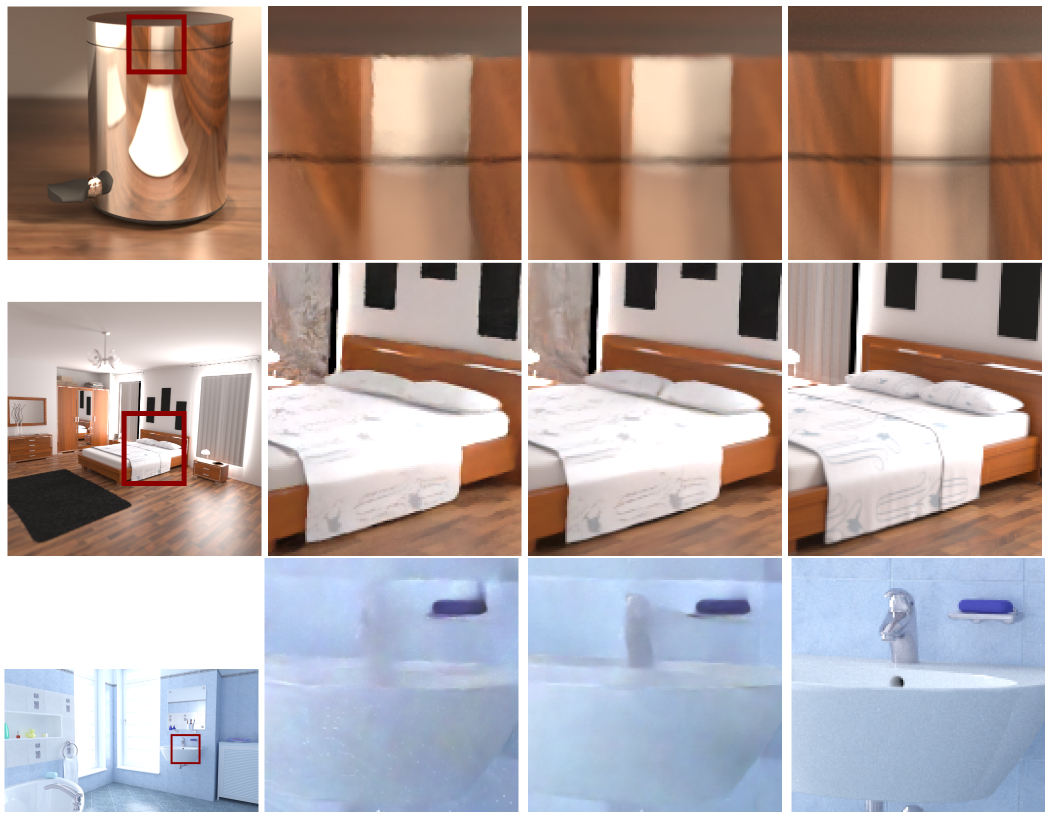


Figure 15: Additional results to Section 4.2

## M KPAL Denoiser Results

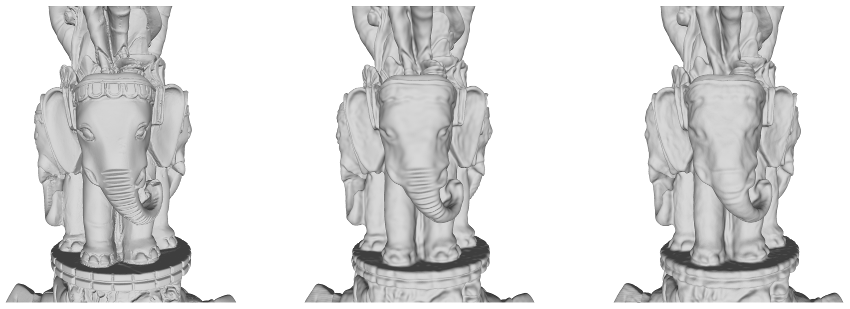
In order to factor out the potential effects of kernel prediction to our denoising experiments, we utilized a direct prediction version of the state-of-the-art Monte-Carlo denoiser presented in [38]. In this section we present results of training a METASIN version of the original kernel prediction network. The improvement over the baseline RELU network remains significant in this experiment across all metrics. An interesting observation from this study was the significantly reduced effect of KD-Bootstrapping, which we eventually decided not to employ to produce the results in Table 15. We hypothesize that the lesser effect of KD-Bootstrapping in this case can be attributed to kernel prediction, which however requires further investigation.

| Metric | KPAL | METASIN-10 (Glorot) |
|---|---|---|
| SMAPE | 3.098 | **2.966 (-4.26%)** |
| flip_loss | 0.934 | **0.893 (-4.39%)** |
| 1-MS_SSIM | 2.893 | **2.622 (-9.37%)** |
| 1-SSIM | 6.227 | **5.934 (-4.71%)** |

Table 15: Comparison of the state-of-the-art kernel predicting denoiser [38] and its significantly improved METASIN version.

## N Failure Cases

While our preliminary experiments on using METASIN activations with MLPs generally resulted in improvements over corresponding baselines, we also observed some failure cases. A particular experiment where the tested METASIN network did not perform well was on signed distance field (SDF) overfitting. In this case, the SIREN reproduction of the ground-truth SDF is better at fine scale details than the corresponding METASIN reproduction. We hypothesize that this discrepancy in performance is due to the gradient loss that SIREN employs and the METASIN network lacks due to not being continuously differentiable. We defer further investigation of this limitation to future work.



(a) GT (b) SIREN (c) $f_j = \mathcal{U}(1, 35)$

Figure 16: SDF overfitting on *Thai Statue* [1]. Here the SIREN network produces a better representation of the underlying 3D shape than a comparable METASIN activated MLP.