# A SUPPLEMENTARY MATERIAL

## A.1 PROOF OF LIPSCHITZ BOUND

As stated in the main paper, for activation functions described by the expression in Eq. (3), the local Lipschitz constant is bounded by $\Theta(\delta(a)+\nabla\delta(a))$ in the neighborhood around an input $a \in \mathbb{R}$, when $\delta$ is differentiable. We demonstrate this by deriving the lower and upper bounds of the derivative of $\sin(x) \cdot \delta(x)$ and showing equality between the two. For the proof, we will assume that $\delta$ is differentiable across all points barring a measure zero set, which means in practice, we are not concerned with singularities in $\nabla\delta$ since they are highly unlikely to occur.

*Proof.* Let $L(a)$ be the local Lipschitz constant of a nonlinearity around an input point $a$. We are interested in $L$ evaluated within a neighborhood $N$ of $a$ as it is a good indicator of if the function oscillates too much for any given input. Ideally, $L$ would grow slowly. Note that bounding $L$ is tantamount to determining the magnitude of the derivative of the nonlinearity.

$$L(a) = \max_{x \in N(a)} |\nabla(\sin(x) \cdot \delta(x))| \tag{9}$$

$$= \max_{x \in N(a)} |\cos(x) \cdot \delta(x) + \sin(x) \cdot \nabla\delta(x)| \tag{10}$$

$$\leq \max_{x \in N(a)} |\cos(x) \cdot \delta(x)| + |\sin(x) \cdot \nabla\delta(x)| \tag{11}$$

Since both $\cos(x)$ and $\sin(x)$ are bounded by $[-1, 1]$, we can write

$$L(a) \leq \max_{x \in N(a)} |\delta(x)| + |\nabla\delta(x)| \tag{12}$$

Note that the absolute value operator simply applies a constant factor to an input and that the derivative of a differentiable function is bounded by a constant factor over a sufficiently small $N(a)$. Therefore, we conclude that $L(a) \in \mathcal{O}(\delta(a) + \nabla\delta(a))$.

Likewise, we wish to determine the lower bound. Since both $\cos(x)$ and $\sin(x)$ are bounded by $[-1, 1]$, we can assign them to the constant values $c_1$ and $c_2$, respectively.

$$L(a) = \max_{x \in N(a)} |\cos(x) \cdot \delta(x) + \sin(x) \cdot \nabla\delta(x)| \tag{13}$$

$$= \max_{x \in N(a)} |c_1 \cdot \delta(x) + c_2 \cdot \nabla\delta(x)| \tag{14}$$

In the worst-case scenario where either $c_1$ or $c_2$ is 0, we have that

$$L(a) \geq \max\ (|c_1 \cdot \delta(a)|, |c_2 \cdot \nabla\delta(a)|) \tag{15}$$

We note that event that this occurs with a probability measure of 0, but lower bounds all other scenarios. Note that $\max(a, b) \in \mathcal{O}(a+b)$. Therefore, it implies $L(a) \in \Omega(\delta(a) + \nabla\delta(a))$. Because both the lower and upper bounds are equal, the Lipschitz constant $L$ evaluated near a point $a$ is in $\Theta(\delta(a) + \nabla\delta(a))$.

□

## A.2 PROOF OF RELATION BETWEEN FOURIER TRANSFORM OF AN IMAGE AND ITS GRADIENT

The proof is adapted from Gonzalez (2013).

*Proof.* Let $f(t)$ be a signal with Fourier transform $F(\omega)$. In the frequency domain, the signal $f(t)$ can be represented as follows:

$$f(t) = \frac{1}{2\pi} \int_{-\infty}^{\infty} F(\omega) e^{i\omega t} d\omega \tag{16}$$

Applying the gradient operator to $f(t)$, we obtain:

$$\nabla_t f(t) = \nabla_t \left( \frac{1}{2\pi} \int_{-\infty}^{\infty} F(\omega) e^{i\omega t} d\omega \right) = \frac{1}{2\pi} \int_{-\infty}^{\infty} i\omega F(\omega) e^{i\omega t} d\omega \tag{17}$$

Therefore, the Fourier transform of $\nabla f(t)$ is $i\omega F(\omega)$, establishing the desired relationship between the Fourier transform of a signal and its gradient. This implies that knowing the frequency representation of a signal is enough to deduce the frequency representation of the gradient of that signal. □

### A.3 Activations

As can be seen in Fig. 8, the outputs of the linear layers are normally distributed. This phenonemon was originally observed to occur for SIREN, where Sitzmann et al. (2020) claim that this proves how the architecture does not suffer from vanishing or exploding gradients.

The outputs of the activations in each layer cluster around the stationary points of $\sin(x) \cdot \sqrt{|x|}$. Note how in the earlier layers of the network, the activations are spread over a wide domain, but by the later layers, they are concentrated in a few stationary points near $x = 0$. We speculate that this demonstrates "learning"/feature selection, where the network first takes in a wide range of frequencies and then aggressively distills it into the most important components.

#### A.3.1 Activation Distributions for Other $\delta$'s

We include the distribution of activations for $\delta(x) = \log(|x|)$ (Fig. 9) and $\delta(x) = \arctan(x)$ (Fig. 10). When the $\delta$ has a large span, these activations can peak at many values, and therefore be somewhat representative of the magnitude of the input that went into it. For example, in Fig. 9, an input into a neuron has many possible "peaks" to choose from, so if it has a higher input magnitude, it will likely be mapped to a higher output magnitude. In Fig. 10, it's mostly going to be either 0 or 1.04, and the exact input that went into it would be very hard to recover. This is because $\sin(x) \cdot \arctan(x)$ has distinct stationary points near the origin, which then all converge to constant values sufficiently far from 0. However, the stationary points for $\sin(x) \cdot \log(|x|)$ keep growing without bound.

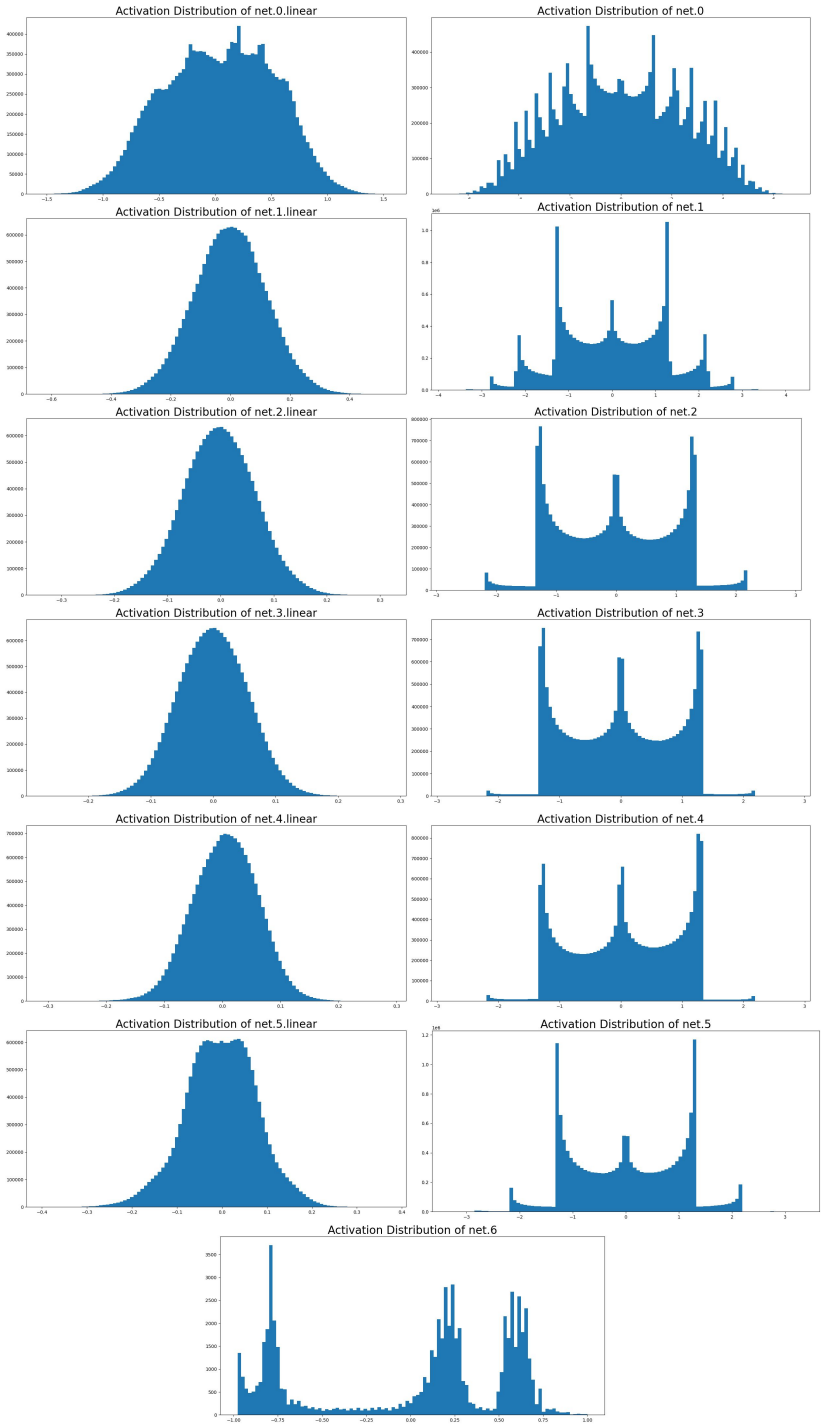


Figure 8: Distribution of activations of each layer of a 5-layer SPDER with $\delta(x) = \sqrt{|x|}$ trained on `skimage.data.camera()`. The output of the linear layers are on the left, and the outputs from the neurons in each layer are on the right, which peak at the stationary points of $\sin(x) \cdot \sqrt{|x|}$ (notice their resemblance to a spider, the architecture's namesake). The activations of the final output are shown at the bottom.

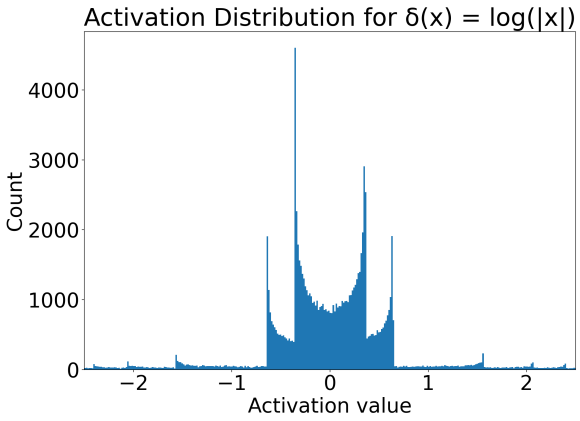


Figure 9: The activation values distinctly peak at local extrema of $\sin(x) \cdot \log(|x|)$ at $\pm 0.36$, $\pm 0.64$, $\pm 1.56$, $\pm 2.06$, etc.

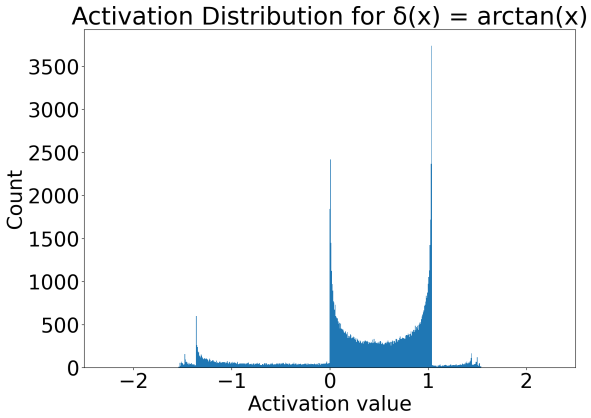


Figure 10: The activation visibly peaks at 0 and 1.04 here, although the stationary points of $\arctan(x) \cdot \sin(x)$ technically approach $-\frac{\pi}{2}$ and $\frac{\pi}{2}$ in either limit. This is because the stationary points nearest $x = 0$, where most of the inputs to neurons are, occur at 0 and 1.04. Beyond that, they're distributed extremely close to $-\frac{\pi}{2}$ and $\frac{\pi}{2}$. See Fig. 2 for a visual of the activation.

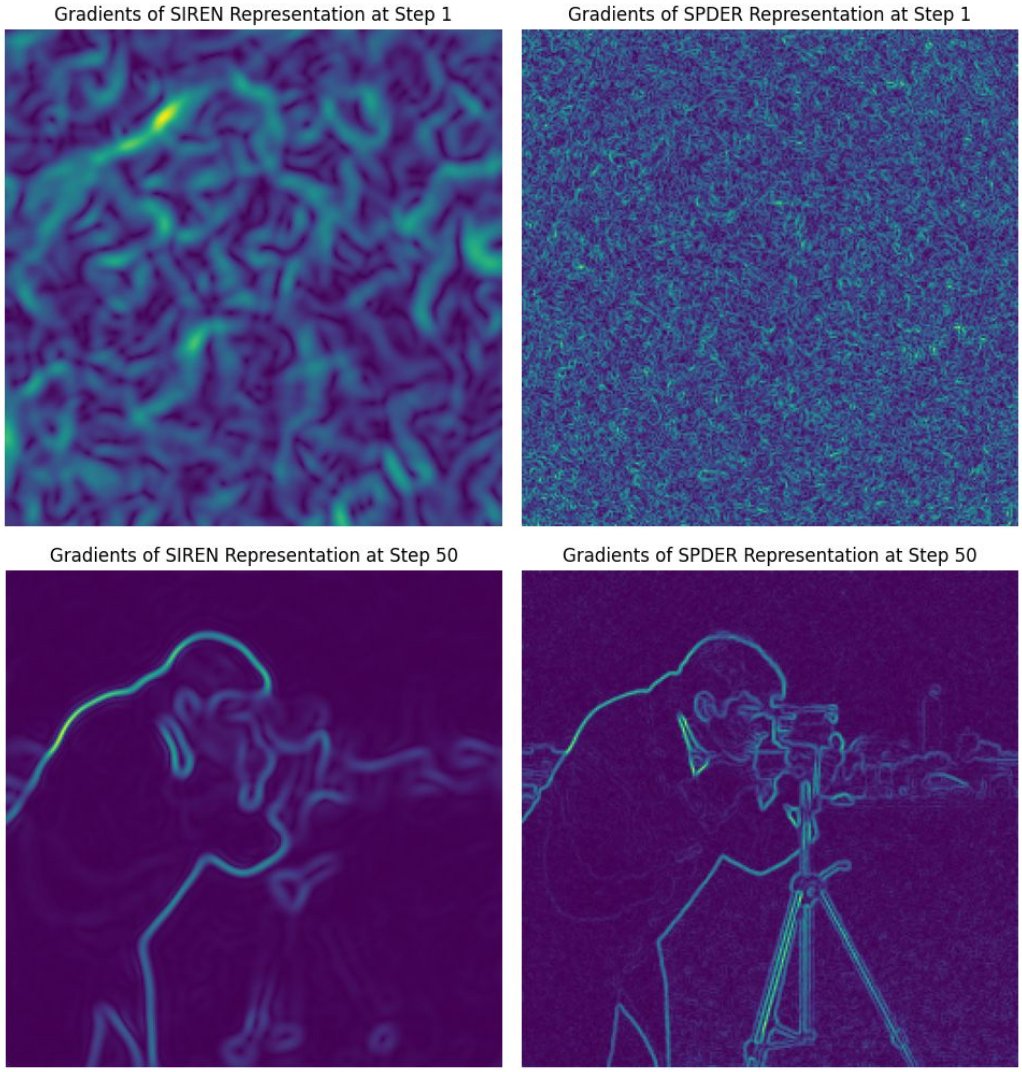


Figure 11: The gradients of the representations after training on the image from `skimage.data.camera()` for a *single* step are shown (top). Interestingly, using only a sine activation (top left) yields a few, thick gradients in the beginning. Adding a damping factor (top right) enables the representation to start off with hundreds of small strands which can presumably align around detail at any frequency much more accurately. We believe this is because of the semiperiodic nature of the activation function, where a *continuum* of frequencies can be encoded by a single neuron, whereas sinusoidals encode some number of *discrete* frequencies.

### A.4 Image Gradient Representation

Visuals are in Fig. 11 and Fig. 12. The intensity of each pixel in the resulting gradient image corresponds to how much a small change in each input coordinate value would affect the corresponding pixel prediction. Areas such as edges and other high-frequency components exhibit high gradient values.

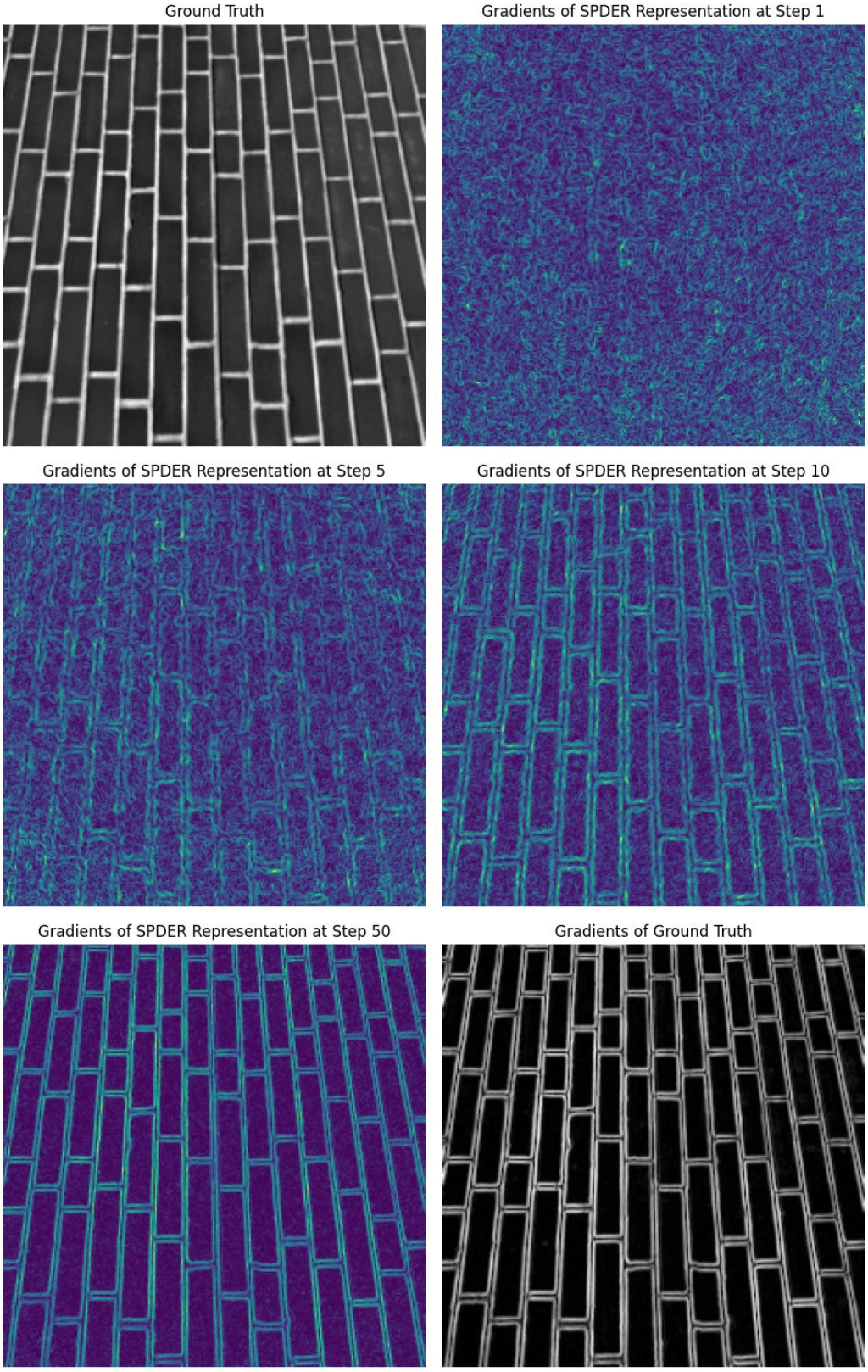


Figure 12: Gradients of SPDER representation of the image from `skimage.data.brick()` over various training steps. Note how SPDER can encode straight edges (low frequencies) incredibly well.

| Metric | Dataset | Super-resolution Factor (SRF) | | | | | |
|---|---|---|---|---|---|---|---|
| | | 2x | | 4x | | 8x | |
| | | SIREN | SPDER | SIREN | SPDER | SIREN | SPDER |
| MSE (↓) | DIV2K | 2e-2±8e-3 | 6e-3±3e-3 | 2e-2±9e-3 | 7e-3±3e-3 | 2e-2±9e-3 | 8e-3±3e-3 |
| | ffhq0 | 9e-3±4e-3 | 3e-3±1e-3 | 8e-3±3e-3 | 3e-3±1e-3 | 9e-3±3e-3 | 3e-3±1e-3 |
| | ffhq1 | 9e-3±5e-3 | 4e-3±3e-3 | 8e-3±5e-3 | 3e-3±3e-3 | 8e-3±5e-3 | 4e-3±3e-3 |
| | ffhq2 | 1e-2±8e-3 | 4e-3±2e-3 | 1e-2±8e-3 | 3e-3±1e-3 | 1e-2±8e-3 | 3e-3±2e-3 |
| | Flickr2K | 4e-2±3e-3 | 1e-2±9e-3 | 4e-2±3e-2 | 2e-2±1e-2 | 4e-2±3e-3 | 2e-2±1e-2 |
| PSNR (↑) | DIV2K | 23.2±1.9 | 29.4±2.6 | 22.8±2.0 | 28.0±2.4 | 22.8±2.0 | 27.7±2.3 |
| | ffhq0 | 26.7±1.6 | 31.6±2.0 | 27.0±1.6 | 32.2±2.0 | 27.0±1.6 | 31.8±1.9 |
| | ffhq1 | 27.3±2.3 | 31.2±2.5 | 27.8±2.4 | 31.9±2.5 | 27.7±2.4 | 31.4±2.6 |
| | ffhq2 | 25.3±2.7 | 30.5±2.1 | 25.8±2.7 | 31.5±1.9 | 25.7±2.7 | 31.0±2.0 |
| | Flickr2K | 23.2±2.0 | 25.6±3.0 | 20.8±2.7 | 25.1±2.8 | 20.8±2.7 | 24.9±2.8 |
| $\rho_{AG}$ (↑) | DIV2K | 0.99165±6e-3 | 0.99751±3e-3 | 0.99110±6e-3 | 0.99712±3e-3 | 0.99108±6e-3 | 0.99702±3e-3 |
| | ffhq0 | 0.99799±1e-3 | 0.99942±4e-4 | 0.99823±1e-5 | 0.99956±3e-4 | 0.99819±1e-3 | 0.99953±3e-4 |
| | ffhq1 | 0.99835±8e-4 | 0.99949±2e-4 | 0.99856±7e-4 | 0.99962±2e-4 | 0.99852±7e-4 | 0.99958±2e-4 |
| | ffhq2 | 0.99775±1e-3 | 0.99920±5e-4 | 0.99812±1e-3 | 0.99945±3e-4 | 0.99807±1e-3 | 0.99941±3e-4 |
| | Flickr2K | 0.96332±3e-2 | 0.98891±1e-2 | 0.96324±4e-2 | 0.98825±1e-2 | 0.96320±4e-2 | 0.98797±1e-2 |

Table 3: **Super-resolution metrics with standard deviations over SRFs of 2×, 4×, and 8×.** The base resolution was 512x512, and the super-resolutions were at 1024x1024, 2048x2048, and 4196x4196, respectively. We chose three different subsets of the Flickr-Faces-HQ dataset (each corresponding to a different category and, therefore, distribution) denoted as ffhq0, ffhq1, and ffhq2. As stated before, by simply applying a damping factor, the super-resolution quality goes up significantly. Note how the PSNRs are ∼5 points higher compared to SIREN in each subcategory. For implementation details, see A.9.4.

### A.5 Super-resolution

In this section, we wish to highlight one application of SPDER's success in capturing the frequency structure: generating higher resolution detail **without the need for priors or meta-learning**.

We noticed that when interpolating from a base resolution of 512x512 or higher, no noise visible to the naked eye is introduced into any segment of the super-resolved reconstruction at *any* super-resolution factor. At base resolutions below that, some noise is visible when interpolating by 4× or more, presumably because the SPDER has too few points to meaningfully learn the frequency distribution. We note this is still impressive, given that SPDER networks have no outside prior on pixel distributions.

#### A.5.1 Results

We include quantitative results of the comparison between SPDER and SIREN in Table 3. Interestingly, SPDER can superresolve to a factor of 8× (64× pixels) and still have high reconstruction quality consistently. Most of the PSNRs of SPDER's super-resolutions are above 30, indicating good quality. We once again highlight how simply applying a damping factor to a nonlinearity boosts performance noticeably.

#### A.5.2 Examples

Visuals are in Fig. 13 and Fig. 14.


Figure 13: Image from `skimage.data.astronaut()` at 512x512 ground truth (top) and SPDER super-resolution of it to 4096x4096 (center). (Bottom, left to right) The detail around the eye is shown at the 512x512 ground truth, bicubic interpolation to 4096x4096, and SPDER's super-resolution to 4096x4096, respectively. Note how compared to bicubic, SPDER is visually sharper and traces out the eyelids better.


Figure 14: Image from `skimage.data.coffee()` at ground truth resolution 256x256 (top) and SPDER super-resolution to 1024x1024 (center). Edge of cup for ground truth (bottom left) and SPDER super-resolution (bottom right) are also shown.

| Metric | UF | Architecture | |
|---|---|---|---|
| | | SIREN | SPDER |
| MSE (↓) | 2x | 0.00719±0.014 | **0.00522**±0.012 |
| | 4x | 0.00959±0.017 | **0.00755**±0.015 |
| | 8x | 0.00959±0.017 | **0.00874**±0.017 |

Table 4: **Audio interpolation results: mean-squared error over various audio upsampling factors (UFs) with standard deviations over UFs of 2×, 4×, and 8×.** Across all upsampling factors, SPDER's mean reconstruction error is lower than that of SIREN's.

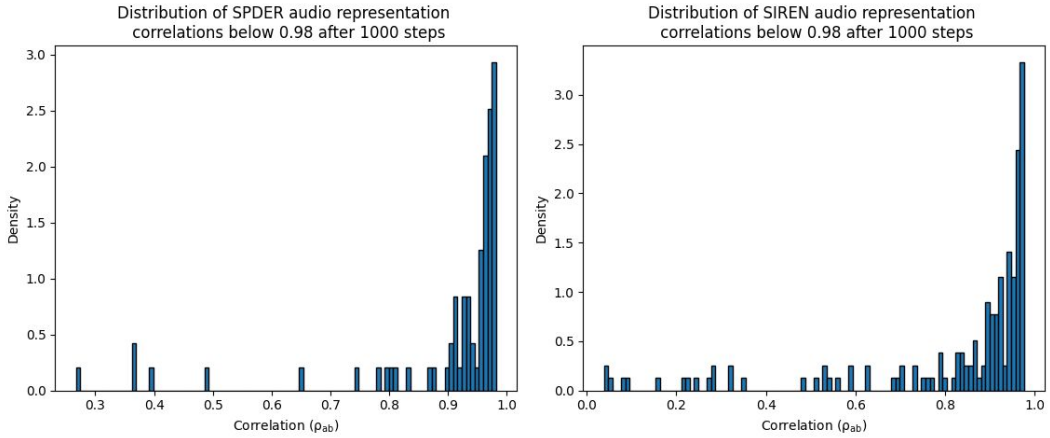


Figure 15: **Distribution of $\rho_{ab}$'s below 0.98 for SPDER and SIREN audio representations.** We drop the top 2 out of 100 bins for both SPDER and SIREN (dropping 89.4% and 81.4% of the total count, respectively) to make these bars visible. Note how SPDER has a noticeably sparser left tail, meaning it can represent many challenging examples SIREN cannot.

## A.6 Audio Representation and Interpolation

### A.6.1 Interpolation

Here, we show SPDER's metrics on reconstructing audio clips even after being trained on a small fraction of the entire clip. This is promising because it demonstrates that applying damping indeed preserves the frequency structure not just for 2D images but 1D audio better than without having it.

Quantitative results are in Table 4. We train an INR on an 8× downsampled version of an entire ground truth audio clip. The model then has to predict the values at the upsampling factors (UFs) of 2×, 4×, and 8×, where the UF is how many times larger the inference sample is than the training one. For example, the 8× upsampled version is simply the ground truth (GT), 4× UF corresponds to GT downsampled by 2×, and 2× UF is GT downsampled by 4×. See further implementation details in A.9.6.

### A.6.2 Audio Discussion

We observed that in both SIREN and SPDER, the vast majority of reconstructions had a high correlation with the ground truth, with the peak in the 0.99-1.00 bin. Both architectures also had a handful of pathological samples they were not able to reconstruct accurately. However, the distribution of $\rho_{AG}$ for SIREN has a noticeably fatter left tail, meaning it has much more difficulty representing some samples that SPDER has no problem with (Fig. 15).

In general, the performance on audio was strongly impacted by a small percentage of pathological (highly noisy) outliers in our dataset. We chose *not* to exclude these from our results to keep the data consistent. However, in the real world, assuming the audio clips are intelligible and not full of noise,

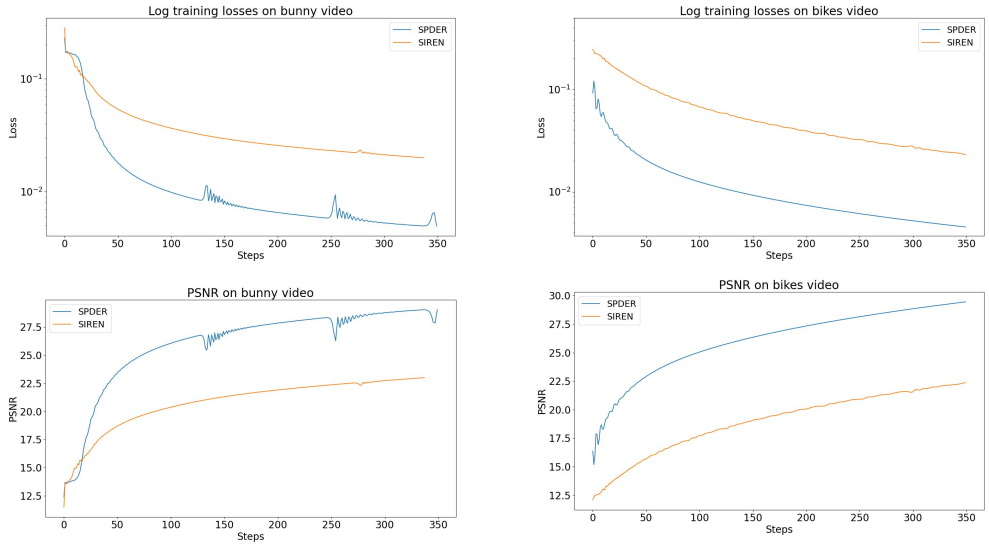


Figure 16: **Log training loss and PSNR curves for two videos for SPDER and SIREN.** Note how SPDER's PSNRs are consistently $\sim$9 points above that of SIREN's (bottom left and right). This illustrates how the simple idea of damping can succeed on non-spatial features (i.e. video frames). There is a performance difference between the two examples because the bunny video has 20% more frames than the bikes video.

SPDER would perform even better than what we present in these results. Training for more steps or using a larger network would also likely lessen the impact of these outliers.

We acknowledge that, ideally, SPDER should not have problems with any samples (however anomalous). Studying these challenging samples may be the basis for future work.

### A.7 Video Representation and Frame Interpolation

First, we trained a SPDER on a portion of Big Buck Bunny, a standard video used in computer vision research. SPDER achieved a peak PSNR of 29.74 within 450 steps, indicating good quality reconstruction. Note how SPDER's representation also serves a lossy compression scheme because the network has $\sim$12 million parameters, but the original video has $\sim$20 million. Second, we trained a 256x256 resolution clip from `skimage.data.bikes()` and it achieved a PSNR of 30.9 within 450 steps. The performances are plotted in Fig. 16.

For the frame interpolation experiments, we trained on the bikes video at 30 FPS, and then asked the model to predict what it would look it if it had twice as many frames (i.e. 60 FPS).

#### A.7.1 Video Representation Examples

The examples, from `skvideo.datasets.bikes()`, are in Fig. 17.

#### A.7.2 Frame Interpolation Examples

Examples are in Fig. 18 and Fig. 19.

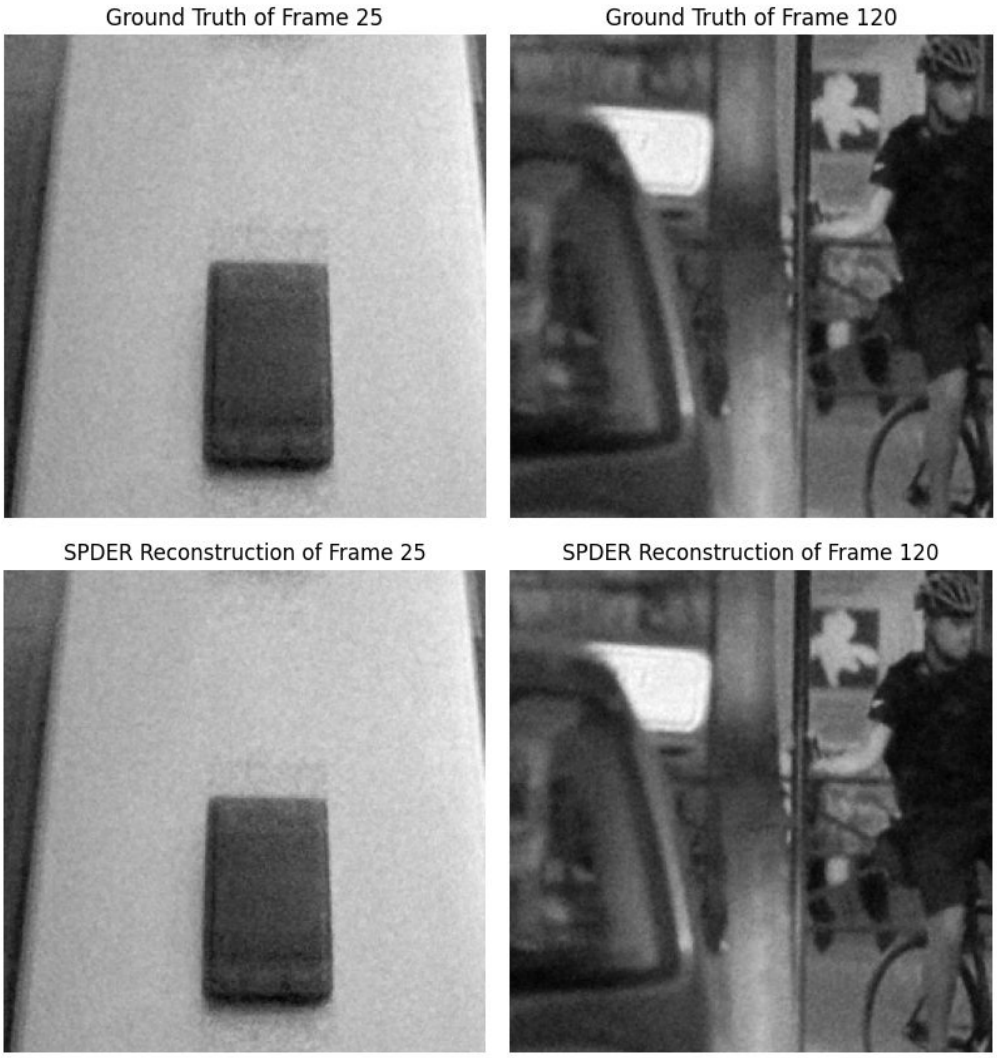


Figure 17: This 250-frame video from `skvideo.dataset.bikes()` is fairly chaotic with many moving parts, yet SPDER is still able to capture it without any artifacts. Here, we chose two frames with overall low and high frequencies (left and right, respectively) to contrast. SPDER evidently has no bias and can encode both accurately.

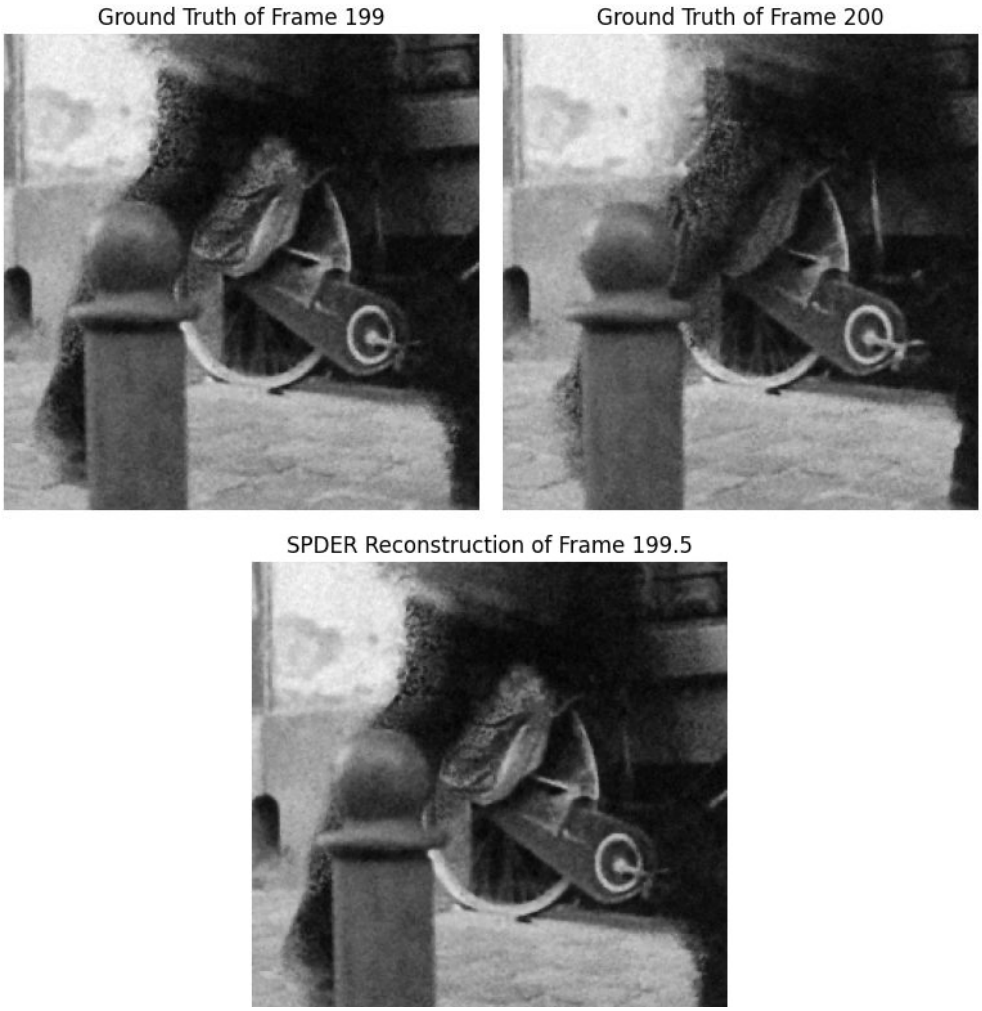


Figure 18: We selected two consecutive frames where movement would be as perceptible as possible. We can see a person's leg move partially behind a pole in the ground truth frames (top left and top right). SPDER's interpolation for the intermediate frame is indeed plausible and doesn't include any visual distortions not already present in the ground truth.

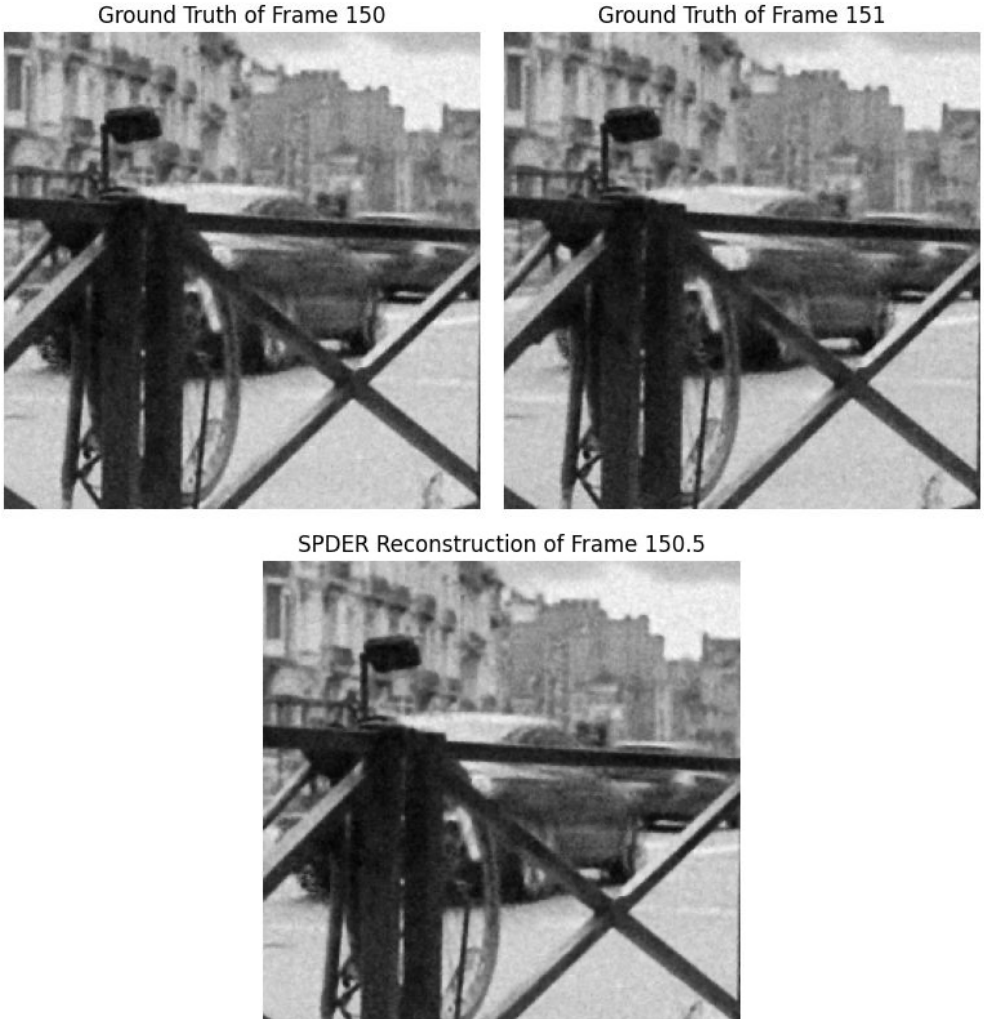


Figure 19: We selected two relatively still consecutive frames and demonstrate how SPDER's interpolation does not seem visually grainier or blurrier (i.e., no higher or lower frequencies introduced) than in frames from the ground truth distribution.

### A.8 COMMENTS ON INSTANT-NGP

Given the hierarchical setup and CUDA C++-tailored implementation, Instant-NGP has more complexity and a significantly large overhead compared to SPDER. We tried running Instant-NGP on several machines, but due to insufficient system requirements, all but a machine with 8 Quadro RTX 6000's (costing over $20,000) failed to run it. Although it offers an impressive speedup compared to other architectures, we found the system requirements to be quite prohibitive compared to using a simple model like SPDER.

That being said, it is also much more efficient and extremely useful for most high-information media. For example, it was much faster than SPDER during training, taking about $\sim$5 seconds for 500 steps on a single 256x256 image, whereas SPDER took $\sim$15 seconds on the same machine. Likewise, it allows for training on objects such as gigapixel images within minutes, which would otherwise take a few hours with the current iteration of SPDER.

To the best of our understanding, Instant-NGP learns a piecewise-linear approximation of the signal and is prone to pseudo-random hash collisions (Müller et al., 2022), so although it achieves very low loss, it technically cannot model the high-frequency detail as well as SPDER. We were unable to retrieve its reconstructed image and therefore could not calculate $\rho_{AG}$ or use it for super-resolution, so we emphasize that any claims we make regarding Instant-NGP's frequency alignment are purely based on our theoretical understanding of it.

We believe that Instant-NGP attempts to overcome spectral bias by having many learned parametric encodings which are responsible for a very small receptive field in the image. Each field has relatively low complexity, which means they can get away with using augmentations like ReLU and bilinear interpolation to approximate the signal piecewise-linearly. However, in SPDER, each neuron has the receptive field of the *entire image*, which means in order to represent it well, it is forced to learn the true frequency structure. To the best of our knowledge, our architecture is the only one that naturally learns the true frequency structure of an image and overcomes spectral bias—notably without hardcoded inductive biases like hierarchies. That being said, we believe combining a hierarchical approach with SPDER is the next reasonable step.

### A.9 REPRODUCIBILITY

By default, we use a 5-layer network with 256 neurons in each layer and a learning rate of $1 \times 10^{-4}$. Input and output values are scaled to be in $[-1, 1]$. As we wish to completely "overfit" to the training data, we use full batch gradient descent (i.e., we do not want the noise from SGD) on each object. We discovered that clamping the input of $\sin(x) \cdot \sqrt{|x|}$ to a tiny value (less than $1 \times 10^{-30}$) avoided division by zero errors on the GPU while maintaining overall functionality. For any resizing operations, we use PyTorch's built-in `torchvision.transforms.Resize` module which uses bilinear interpolation under the hood.

We used two GeForce GTX 1080 Ti's with 12 GB of memory each. We note that all SPDER experiments can be run on personal laptops and require no tailored hardware.

Please refer to the full source code for reproducing the experiments in the supplementary material.

#### A.9.1 WEIGHT INITIALIZATION

We followed the weight initialization scheme from Sitzmann et al. (2020). Weights are initialized from $\mathcal{U}\left(-\sqrt{\frac{6}{\text{fan\_in}}}, \sqrt{\frac{6}{\text{fan\_in}}}\right)$, where fan_in is the number of inputs to a layer. They claimed this leads to the input of each activation being normally distributed and keeps gradients stable, which we indeed observed in SPDER (Fig. 8).

Also consistent with their implementation, we multiply the input into each activation by a predetermined $\omega_0$, which is always 30.

#### A.9.2 IMAGE REPRESENTATION

Recall that for INRs, the training data are the pixels from a *single* image; the model's goal is to overfit it. The training process can be viewed as an "encoding" step, where the data from an image is loaded

onto the weights of the model. The "decoding" step is simply a single forward pass of the model, where the information from the weights is mapped onto a prediction at each pixel coordinate.

For these experiments, we sampled images from DIV2K, a high-quality super-resolution dataset. Each image was resized to a resolution of 256x256 and fed into an untrained network, which would then train on it for a fixed number of steps. For simplicity, only one channel of each image was included, but note that representing all three is trivial. The loss, peak signal-to-noise ratio (PSNR), and $\rho_{AG}$ between the overfit model's representation after training and ground truth were recorded for training steps 25, 100, and 500. Loss is the mean-squared error between the model's prediction compared to the ground truth over each pixel. Recall that the values come from a scale of $[-1, 1]$ (A.13). PSNR is calculated through Eq. (23). $\rho_{AG}$ comes from Eq. (5) and is simply the cosine similarity between the amplitude spectrum of the model's reconstruction and the ground truth image.

For Instant-NGP, we had to use a much more high-end machine (A.8) to collect data. Note that for ReLU with FFN, the method requires manual tuning of a scale hyperparameter, so perhaps a configuration better than a standard Gaussian exists, but it was not apparent to us even after several attempts. SPDER, on the other hand, requires no such manual tuning for different datasets.

### A.9.3 Gradient Representation

For gradient representation, we overfit a network to an image using the same hyperparameters as before. The gradient of the reconstructed image is then computed. Specifically, we use the PyTorch function `torch.autograd.grad` to calculate the gradient of the output of the model (i.e., the reconstructed pixels) with respect to the coordinate inputs.

### A.9.4 Super-resolution

For these experiments, we utilized three datasets – DIV2K, Flickr2K, and Flickr-Faces-HQ. DIV2K, a standard benchmark for image super-resolution, comprises 2K resolution RGB images. Flickr2K is a diverse dataset of high-resolution images sourced from Flickr, widely used for training super-resolution models. Lastly, Flickr-Faces-HQ (FFHQ) is a dataset of high-quality PNG images at 1024x1024 resolution, employed to test model performance on detailed, structured imagery like human faces. We test images from each dataset on three super-resolution factors (SRFs) – 2, 4, and 8. For each combination of SRF, dataset, and architecture, we use $N = 12$ images from each dataset.

For training on each dataset, we first sample an image, select a single channel, and then resize it down to 512x512 (the base resolution). We overfit the model (either SIREN or SPDER) to the resized image by training it for 100 steps (by when both models' reconstructions reasonably converge). Using the trained model overfit to the base resolution version of the image, we then ask the model to predict what the image would look like at a resolution that is $2\times$, $4\times$, or $8\times$ that of the base resolution. Recall this is possible because INRs learn a *continuous* representation for an object, and can therefore be queried at any input coordinate value within a reasonable domain. To compute scores on our metrics, we take the original image along the single channel and (if needed) resize it at the given super-resolution to get the ground truth. We calculate the metric between the model's prediction and the ground truth at the given super-resolution, using the same procedure as for that of image representation.

### A.9.5 Audio Representation

Consistent with the implementation by Sitzmann et al. (2020), we 1) scale the input grid to have bounds $[-100, 100]$ instead of $[-1, 1]$, which is tantamount to scaling the weights of the input layer by 100, and 2) use a learning rate of $5 \times 10^{-5}$.

Each training sample was cropped to the first 7 seconds of clips from ESC-50, a labeled collection of 2000 environmental audio recordings, and then trained on for 1000 steps.

### A.9.6 Audio Interpolation

For each ground truth, we sample a clip from ESC-50 and crop it to the first 4 seconds. We train the network (either SIREN or SPDER) on an $8\times$ downsampled version of this ground truth clip for 250 steps. For inference, we then take the original ground truth clip and downsample it by $4\times$, $2\times$, and

$1\times$. Note the overfit network will have seen only half of the first set of values, a quarter of the second, and an eighth of the third during training. The model then outputs a prediction for the ground truth audio clips across these resolutions, and the MSE is recorded. For each combination of architecture and resolution, we use $N = 30$.

Unless stated otherwise, the rest of the implementation is the same as that of for audio representation.

### A.9.7 VIDEO REPRESENTATION

We used a 12-layer 1024-neuron wide network with a learning rate of $5 \times 10^{-6}$ for our video experiments. Each architecture (SIREN and SPDER) was trained for at least 350 steps, and the loss was recorded at each step. To speed up training, we selected a single channel from each video and center-cropped then resized to $256\times256$. Training on a single video for SPDER took $\sim$6 hours on our setup with 24 GB of compute.

The first video we trained on was from `skvideo.datasets.bigbuckbunny()`, a standard video used in computer vision research. The first 300 frames were kept. The second video was from `skvideo.datasets.bikes()`. The first 250 out of 300 total frames were kept.

### A.9.8 VIDEO FRAME INTERPOLATION

We used the SPDER trained on the 250-frame video from `skvideo.datasets.bikes()` from the representation experiments. While the original video was shot in 30 FPS, we asked the SPDER to predict what it would look like at 60 FPS by inputting the same coordinate grid but $2\times$ denser along the frame dimension. We then visualized the results.

Unless stated otherwise, the rest of the implementation is the same as that of for video representation.

## A.10 LIMITATIONS

- SPDER shows the most success on tasks where the input comes from a coordinate grid. For instance, it converges slower than NeRFs for novel view synthesis, presumably due to the non-affine nature of the view direction input. Future work can explore preprocessing approaches to aptly project such inputs onto a coordinate space to potentially see the same success as for images.
- When compared to SIREN, SPDER's training is $\sim1.3\times$ slower since it must calculate the damping factor for each input. While storing the gradients does consume $\sim1.5\times$ more memory, this issue can be circumvented through batch processing.
- Determining the optimal $\delta$ for media types other than image, audio, and video may require some experimentation.

## A.11 IMAGE FFT FORMULA

In the main paper, we included the general formula for the Fourier amplitude spectrum of a discrete-time signal for a vector input. However, the Fourier transform can also apply to tensors, which is relevant in the case for images. Here, we include the formula for the Fourier transform of a 2D tensor:

$$\mathcal{F}(u,v) = \frac{1}{MN}\sum_{x=0}^{M-1}\sum_{y=0}^{N-1} f(x,y)e^{-i2\pi\left(\frac{ux}{M}+\frac{vy}{N}\right)} \tag{18}$$

which we used when evaluating the frequency similarity of images. In this formula, $\mathcal{F}(u,v)$ represents the Fourier transform of the 2D array $f(x,y)$ with respect to the spatial frequencies $u$ and $v$. The size of the array is $M$ by $N$. The summation is taken over all possible values of $x$ and $y$ in the array. The exponential term represents the phase shift at each point in the array. Note that the method `numpy.fft.fft2` can be used to calculate this efficiently, so we included this just for reference.

## A.12 ABLATION ON VARIOUS $\delta$'S

We tested various forms of $\delta$ on the image from `skimage.data.camera()` over 250 steps in Fig. 20. The log-loss results are shown. Using a $\delta(x)$ of $\text{ReLU}(x)$ or just $x$ performs extremely poorly,

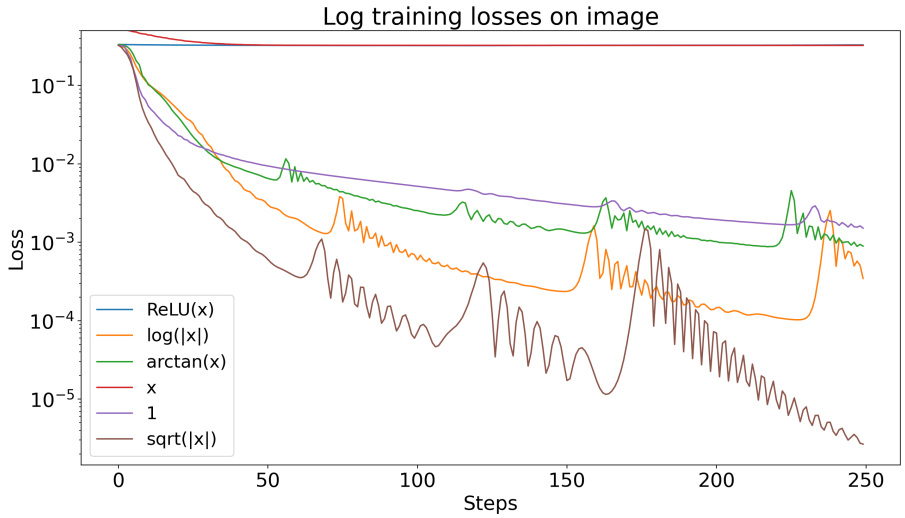


Figure 20: Log training losses on the cameraman image for various $\delta$'s

and the loss curves are cropped off from the top because the losses are so high. The performance of the other $\delta$'s in order from best to worst is $\sqrt{|x|}$, $\log(|x|)$, $\arctan(x)$, and then 1 (which is simply a sinusoidal, used in SIREN).

The loss curves in Fig. 20 for SPDER and SIREN look different from Fig. 1 because we trained on a CPU with no clamping here, but the overall trend remains.

### A.13 Correspondence to 8-bit Scale

In our training setup, inputs and outputs are scaled to be in $[-1, 1]$. The mean-squared error (MSE) loss is calculated on this scale as well. To interpret this on an 8-bit scale (where pixel values are integers, $p$, with $p \in \{0, 1, \ldots, 255\}$), we need to apply the following transformation on an output value $y \in [-1, 1]$:

$$f : y \to p; \quad f(y) = \frac{(y+1) \cdot 255}{2} \tag{19}$$

In practice, however, the bounds of the prediction values slightly diverge from $[-1, 1]$, so the most reliable way to reconstruct images is through normalization:

$$f : y \to p'; \quad f(y) = \frac{(y - y_{min}) \cdot 255}{(y_{max} - y_{min})} \tag{20}$$

If we want to calculate the MSE on the 8-bit pixel scale, we apply this transformation:

$$f : MSE_y \to MSE_p; \quad f(MSE_y) = MSE_y \cdot \left(\frac{255}{2}\right)^2 \tag{21}$$

where we assume $y_{max} - y_{min} \approx 2$. In Table 1, we demonstrate how SPDER has an average training loss of $6.7 \times 10^{-8}$, which corresponds to an MSE of 0.001 pixels$^2$ on an 8-bit pixel scale, so incorrectly predicted bits are extremely rare. No reconstruction at this resolution had any artifacts, according to our observations.

### A.14 PSNR Calculation

The formula for peak signal-to-noise ratio (dB) is given by:

$$\text{PSNR} = 10 \cdot \log_{10}\left(\frac{\text{MAX}_I^2}{\text{MSE}}\right) \tag{22}$$

Note that outputs are scaled from $[-1, 1]$ and have a range of 2. Using the same MSE in training, we can calculate PSNR as follows:

$$10 \cdot \log_{10}\left(\frac{(1-(-1))^2}{\text{MSE}}\right) = 10 \cdot \log_{10}\left(\frac{4}{\text{MSE}}\right) \tag{23}$$

Therefore, PSNR can be directly determined from the training loss (MSE) on each sample. We only include it because of its interpretability.