$(\text{Implicit})^2$: Implicit Layers for Implicit Representations

Zhichun Huang
Carnegie Mellon University
Pittsburgh, PA 15213
zhichunh@cs.cmu.edu

Shaojie Bai
Carnegie Mellon University
Pittsburgh, PA 15213
shaojieb@cs.cmu.edu

J. Zico Kolter
Carnegie Mellon University
Pittsburgh, PA 15213
zkolter@cs.cmu.edu

Abstract

Recent research in deep learning has investigated two very different forms of "implicitness": *implicit representations* model high-frequency data such as images or 3D shapes directly via a low-dimensional neural network (often using e.g., sinusoidal bases or nonlinearities); *implicit layers*, in contrast, refer to techniques where the forward pass of a network is computed via non-linear dynamical systems, such as fixed-point or differential equation solutions, with the backward pass computed via the implicit function theorem. In this work, we demonstrate that these two seemingly orthogonal concepts are remarkably well-suited for each other. In particular, we show that by exploiting fixed-point *implicit layer* to model *implicit representations*, we can substantially improve upon the performance of the conventional explicit-layer-based approach. Additionally, as implicit representation networks are typically trained in large-batch settings, we propose to leverage the property of implicit layers to *amortize* the cost of fixed-point forward/backward passes over training steps – thereby addressing one of the primary challenges with implicit layers (that many iterations are required for the black-box fixed-point solvers). We empirically evaluated our method on learning multiple implicit representations for images, audios, videos, and 3D models, showing that our $(\text{Implicit})^2$ approach substantially improve upon existing models while being both faster to train and much more memory efficient.

1 Introduction

The concept of "implicitness" has recently been applied to various contexts of machine learning research. One particular thread is *implicit representations* [20, 23, 29, 28], which aims to learn a continuous representation of high-frequency, often discretely-measured signals such as large images. Formally, given a spatial-temporal coordinate input $x \in \mathbb{R}^d$, an implicit representation of it is a function $\Phi : \mathbb{R}^d \to \mathcal{C}$, where $\mathcal{C}$ is the space of desired quantity (e.g. color, volume density, distance, etc.) and Φ is usually parameterized with a neural network. Such implicitness has multiple benefits over the conventional discrete (e.g., grid-based) representations; e.g., as Φ is defined on a continuous domain, an implicitly represented input consumes much less storage than the original input; as another example, Φ is differentiable, hence allowing for the computation of higher-order derivatives. However, the training of these Φ networks themselves are usually quite memory-consuming, since we usually deal with very high-resolution images and videos, and training is typically done in full-batch mode (e.g., when training on a batch size of 512×512, a simple 4-layer MLP with 1024 hidden units already requires > 16 GB memory just to store the intermediate activations).

A recent (orthogonal) usage of *implicitness* in deep learning comes from the notion of *implicit layers/models*, where the word is used to characterize model architectures (rather than input representations). The nomenclature comes from the concept of implicit vs. explicit functions: instead of

35th Conference on Neural Information Processing Systems (NeurIPS 2021).

representing a model as an explicit stacking of layers, an implicit model solves a non-linear dynamical system [10] (e.g., ODEs [7, 9] or fixed-points [1, 33]). Importantly, these implicit models decouple the forward and backward passes and analytically differentiate via the implicit function theorem [14]. This enables these models to use *constant training memory*, irrespective of the forward pass trajectory. Moreover, recent works on deep equilibrium models [1] have been shown to be able to achieve results on par with the state-of-the-art explicit models (e.g., Transformers [32]) on very large-scale vision or NLP tasks [1, 2]. However, despite training memory efficiency, these implicit layers are also slower to compute (e.g., by 2-3$\times$), since the solvers are usually iterative in nature and ultimately require many function evaluations.

Despite the success of these "implicit" methods in their respective areas, the two concepts so far have rarely intersected: existing models used for implicit representations are neither *implicit* in nature (since they are still stacked multi-layer explicit operators trained end-to-end in the feedforward manner), nor do they exploit any properties of the implicit functions during training (e.g., implicit differentiation). In this paper, we argue that implicit representations and implicit layers are remarkably well-suited to each other, and their use together can compensate for their aforementioned drawbacks.

Specifically, we propose to combine the best of both worlds by replacing the explicit models used in conventional *implicit representation* learning with implicit layers, whose output is defined by a *fixed-point condition*, thus dubbed $(\text{Implicit})^2$ networks. Formally, let x be the input and F be an (often shallow) layer, the $(\text{Implicit})^2$ approach learns the following implicit representation Φ:

$$\Phi(x) = Wz^\star + b, \text{ where } z^\star = F(z^\star; x) \tag{1}$$

Importantly, we show that these two "implicitness" complement each other well, especially in two important aspects. First, the unique large-batch training scheme of *implicit representations* as well as the forward/backward decoupling properties of *implicit models* permit us to *amortize* the cost of the iterative solver that would otherwise make implicit models slow to train. We show that by fixed-point reuse (in the forward pass) and Jacobian approximation (in the backward pass), training an implicit model on implicit representation tasks requires almost no computation overhead. Second, implicit models allow us to train these high-frequency input tasks with a much lower memory consumption while not losing speed or performance, thus significantly lowering the hardware requirement and computation budget of training implicit representations. We evaluate $(\text{Implicit})^2$ on multiple implicit representations for images, videos, and audios, showing substantial improvement upon existing competitive explicit-model methods like SIREN [28] or MFN [11] using equivalent-sized networks.

2 Background and Preliminaries

2.1 Implicit Neural Representations (INR)

While high-frequency data such as images and scene geometry have been traditionally represented discretely (e.g., pixel/voxel grids or mesh points), recent work has demonstrated the possibility of replacing them with continuous functions parameterized by multi-layer perceptrons (MLPs) [28, 29]. These deep networks have been used to learn differentiable representations of various forms, such as continuous spatial-temporal image representations [5, 8], high-resolution scene representations [27, 23], and volumetric rendering [20, 30], and have been shown to be significantly more compact than the conventional grid-based approaches.

However, training these INR models is not easy. For example, the training of implicit representations is usually conducted in very large batch sizes (e.g., 512×512), which renders these (albeit simple) MLPs very memory-consuming during training. As another example, recent works have shown that typical deep networks with ReLU/tanh/sigmoid non-linearities fail to capture the fine details of an input signal, and resorted to periodic non-linearities [28], Fourier feature encodings [29], or repeated application of non-linear filters (e.g., Gabor wavelets) to the input and then multiply with the features [11]. We introduce below the two latest, highly competitive models that have been developed on this task, which our work will build on (see §3.1).

Sine-activated Network for Implicit Representations (SIREN). To overcome the detrimental effect of traditional non-linearities like ReLU/tanh on modeling fine details and higher-order derivative of the input signals, Sitzmann et al. [28] proposes to use sinusoidal activation functions that allow explicit supervision on any derivatives of the input signal. The resulting MLPs, though simple, have been shown to achieve state-of-the-art performance in representing images, videos, complex

geometries, and more [28]. Formally, given an input x, SIREN learns an implicit representation by the following L-layer MLP:

$$\Phi(x) = W(g_{L-1} \circ g_{L-2} \circ \cdots \circ g_0)(x) + b \tag{2}$$

where g_i is the i-th layer of the network that computes $g_i(h^{[i]}) = \sin(W_i h^{[i]} + b_i)$ with a sine non-linearity, and $h^{[i]}$ denotes the hidden state at i-th layer.

Multiplicative Filter Networks (MFN). More recently, Fathony et al. [11] proposed to forego the SIREN-like design (i.e., MLPs with periodic activation functions) in favor of multiplicative operations that are proven to be similarly expressive. Specifically, at each layer i of the network, an MFN applies a learnable non-linear filter kernel (parameterized by $\theta^{(i)}$) $g(x; \theta^{(i)})$ on the *original input* $x \in \mathbb{R}^n$, and elementwise multiply it with a *linear transformation* of the features; i.e., formally,

$$\Phi(x) = W h^{[L]} + b, \text{ where } h^{[i+1]} = \big(W_i h^{[i]} + b_i\big) \cdot g(x; \theta^{(i)}) \text{ and } h^{[0]} = g(x; \theta^{(0)}) \tag{3}$$

Fathony et al. [11] in particular proposed two instantiations that admit sinusoids or a Gabor wavelet as the filter g, which are called FourierNet and GaborNet, respectively:

$$\begin{aligned} g_{\text{Fourier}}(x; \theta^{(i)}_{\text{Fourier}}) &= \sin(\omega^{(i)} x + \phi^{(i)}) \\ g_{\text{Gabor}}(x; \theta^{(i)}_{\text{Gabor}}) &= \sin(\omega^{(i)} x + \phi^{(i)}) \cdot \exp\big(-(\gamma^{(i)}/2) \cdot (x - \mu^{(i)})^2\big) \end{aligned} \tag{4}$$

where $\theta^{(i)}_{\text{Gabor}} = \{\omega^{(i)} \in \mathbb{R}^n, \phi^{(i)} \in \mathbb{R}, \gamma^{(i)} \in \mathbb{R}, \mu^i \in \mathbb{R}^n\}$ (similar for $\theta^{(i)}_{\text{Fourier}}$), and $(\mu^{(i)}, \gamma^{(i)})$ denotes the mean and scale of the Gabor filter. These models have shown performance on par with or better than the periodic-nonlinearity-based networks like SIREN on a range of tasks like image and video representations.

2.2 Implicit Models

Whereas explicit models hierarchically stack a sequence of L non-linear operators to build a computation graph for their forward/backward passes (where L is often referred to as the *depth* of a model), implicit model instead define the output as a solution to a non-linear dynamical system which in most cases can't be expressed as a fixed computation graph. For example, the Neural ODE model [7] solves an initial value problem ODE that intuitively corresponds to infinitesimal residual layers [13]. Another example is the recent deep equilibrium (DEQ) models [1] , whose forward pass involves computing the fixed point of a shallow layer F.

Importantly, unlike explicit models, these implicit models do not need to store any intermediate activations during training: to produce gradients in the backward pass, we can directly differentiate through their *final outputs* (e.g., by adjoint method for Neural ODEs [7] and implicit differentiation for equilibrium models [1]), regardless of how the forward pass was computed. Therefore, the forward and backward passes of implicit models can rely on independent black-box solvers, and these models consume only a constant training memory. Our work is primarily built on the formulation of deep equilibrium models, which we introduce in detail below.

Deep Equilibrium (DEQ) Models. Given an input x and a layer $F(z; x)$ (e.g., a residual block, or a multi-head self-attention layer [32]), a DEQ model aims to approximate an infinite-level stacking of this layer (i.e. $z^{[i+1]} = F(z^{[i]}, x)$ with $i = 1, \ldots, L$ and $L \to \infty$). To achieve this purpose, Bai et al. [1] advocated for directly solving a fixed-point condition defined by F and x:

$$z^\star = F(z^\star; x) \tag{5}$$

where $z^\star$ is called the equilibrium since, intuitively, stacking another $F(\cdot, x)$ layer on top does not change the activation. To solve for the equilibrium point, prior works have had success with various kinds of advanced solvers, such as Broyden's method [6, 1, 18], and Peaceman-Rachford splitting [24, 33, 25]. Then, in the backward pass, we can produce the gradient by the implicit function theorem [14] directly applied on this equilibrium point $z^\star$, with a loss ℓ:

$$\nabla_{(\cdot)} \ell = \frac{\partial \ell}{\partial z^\star} \left(I - J_F\right)^{-1} \frac{\partial F(z^\star; x)}{\partial (\cdot)} \tag{6}$$

where J_F denotes the Jacobian of F evaluated at $z^\star$. As the inverse of a matrix is expensive to compute, in practice the backward pass is usually formulated as another *linear* fixed-point condition

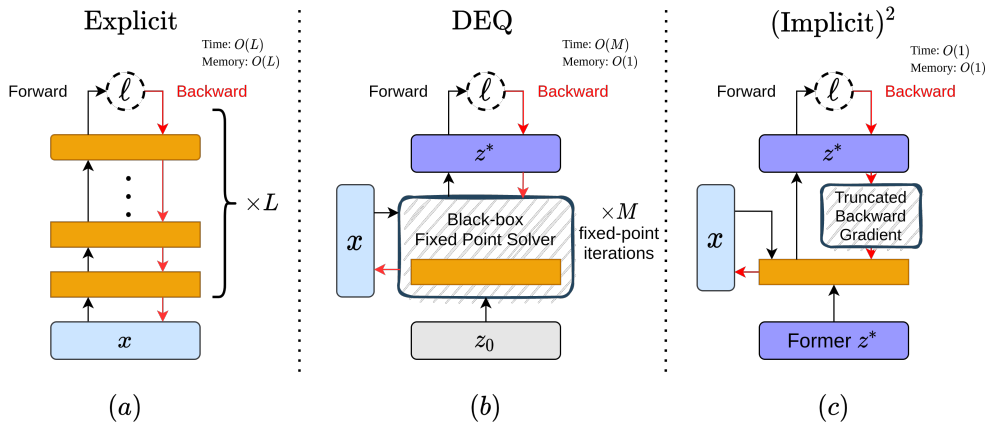

Figure 1: (a) Explicit networks incur $O(L)$ complexity in both time and memory during training. (b) Original DEQ models requires constant memory, yet they take $O(M)$ time in forward/backward passes, where M is the number of fixed-point solver steps. (c) In contrast, an $(\text{Implicit})^2$ Network consumes constant amortized memory *and* time in both forward and backward passes.

in y: $y^\top J_F + \frac{\partial \ell}{\partial z^\star} = y^\top$, which again can be solved via any of the advanced black-box solvers. Thus theoretically, the model only needs to store x, $z^\star$ and $F(z^\star; x)$ in memory at training time, which is equivalent to the memory consumption of merely one F layer. Recent works [1, 2] have demonstrated that DEQ models can scale to high-dimensional tasks and be as competitive as explicit structures like ResNet [13] and Transformers [32] on large-scale computer vision and NLP tasks. However, these implicit models are also slower (typically by $\sim 3\times$) to train, because both forward and backward passes involve numerous iterative steps.

Our work studies the novel combinations of *implicit models* and *implicit representation learning*. Specifically, we show that these two "implicitness" are surprisingly well-suited to each other, as they almost perfectly compensate for each others' drawbacks. We demonstrate that implicit modeling of a simple layer F substantially improves the training speed, memory, *and* performance on implicit representation tasks when compared to the aforementioned state-of-the-art deep explicit networks.

3 $(\text{Implicit})^2$ Networks

As we discussed in §2.1, learning implicit representation for complex signals is often more effective when projecting the input coordinates onto high-frequency bases (e.g. sinusoidal functions), making the designs of these models typically different from deep networks used in more common applications, like image classifications (e.g., ResNets [13]). In this work, we propose to directly build on the designs of these aforementioned state-of-the-art layer architectures by only making minimal modifications, but modeling them as shallow implicit models. We briefly introduce below the two instantiations of our proposed $(\text{Implicit})^2$ approach, which are based on SIREN [28] and MFN [11], respectively, followed by a discussion of how implicit representation tasks can enable us to amortize the cost of training these implicit networks significantly in practice.

3.1 $(\text{Implicit})^2$ Network Architctures

Implicit Sine-activated Networks (iSIREN). Motivated by the success of SIREN, we first propose an *input-injected* variant of SIREN (denoted as *SIREN (input inj.)*) suitable for implicit models, i.e.,

$$F_{\text{SIREN}}(z; x) = \sin(W(z + \sin(Vx)) + Ux + b) \tag{7}$$

where, following prior works on deep equilibrium models [1, 33, 25], the transformed input x is added to the hidden representation z (see Fig. 2). Just like the canonical SIREN layer, any order of derivative of this input-injected SIREN variant is also an input-injected SIREN of the same form. Therefore, the proposed variant inherits the same high-frequency and higher-order differentiability properties as SIREN, and we name the implicit model that solves its fixed-point as *iSIREN*.

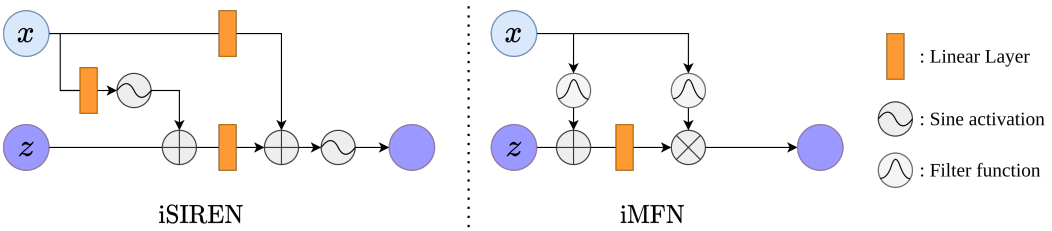

Figure 2: The architecture of one implicit layer $F(z; x)$ in iMFN and iSIREN. Note that when $z = 0$, evaluating $F(z; x)$ once is equivalent to the output of the respective explicit layer.

Implicit Multiplicative Filter Networks (iMFN). Additionally, we introduce a variant based on the recent Multiplicative Filter Networks (MFN) [11]. As MFNs only perform linear transformations on the hidden features, which are multiplied with a non-linear filter function on the original input x, we slightly modify the layer design as follows to introduce an additional input injection:

$$F_{\text{MFN}}(z; x) = (W(z + g(x; \theta_2)) + b) \circ g(x; \theta_1) \tag{8}$$

where g denotes the filter function of choice, such as Gabor or Fourier (g_{Gabor} and g_{Fourier} in Eq. (4)) filters [11]. We call the implicit models defined by this layer iMFN. Moreover, just like the original MFN [11], we can show that the output of iMFN (which is the fixed-point of F_{MFN}) is also a linear combination of the non-linear filter functions (see Appendix B for proof).

We provide visualizations of these two architectural instantiations in Fig. 2. With these variants, we formally introduce the corresponding (Implicit)^2 approaches to implicit representation tasks as follows:

$$\Phi(x) = W' z^\star + b', \text{ where } z^\star = F(z^\star; x), \text{ and } F \in \{F_{\text{MFN}}, F_{\text{SIREN}}\} \tag{9}$$

3.2 Accelerated Training of (Implicit)^2 Networks

Compared to explicit networks, training with implicit models significantly lowers the *memory* budget required (thanks to implicit differentiation). This is especially compelling for training implicit representation tasks, which are often memory-bottlenecked. In this subsection, we describe how one can exploit 1) the large-batch nature of implicit representation training; and 2) the nature of fixed-point optimizations (in both forward and backward), to significantly improve the *speed* of these models as well, making them eventually both more memory-efficient *and* time-efficient than explicit models (while achieving performance as competitive, or better; see §4).

Fixed-point Reuse. Suppose we fix a training step t and let the corresponding network parameter (i.e., weights of the network) be given by $\theta_{(t)}$. Typically, a deep equilibrium model solves for the fixed-point $z^\star_{(t)}$ given an input x with a black-box fixed-point solver starting with an initial guess $z^{[0]}$ (which is usually $\mathbf{0}$ or a random vector in $\mathcal{N}(0, I)$):

$$z^\star_{(t)} = \mathsf{FixedPointSolve}(F, \theta_{(t)}, x, z^{[0]}) \tag{10}$$

where we note that $z^\star_{(t)}$ does not depend at all on this initial guess $z^{[0]}$ of the optimization. Depending on the quality of the initial guess, the optimization process itself can take a different number of steps. For example, in the unlikely circumstance where we guessed $z^{[0]}$ to be $z^\star_{(t)}$, the fixed-point solvers will directly converge at iteration 0. However, in practice, this property is rarely used, as given a complex high-dimensional input x (e.g., a high-resolution image), it is hard to know (or learn) what makes a good initial guess.

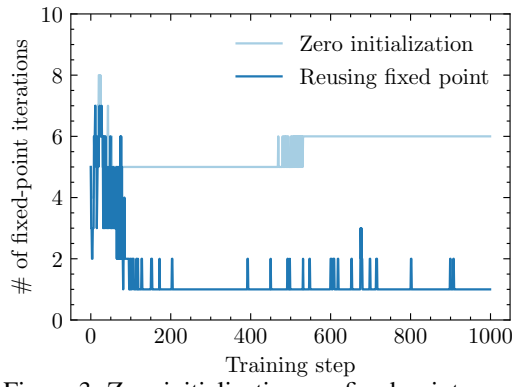

Figure 3: Zero initialization *v.s.* fixed-point reuse in terms of required steps to converge. Further details of this experiment can be found in §4.1.

Therefore, these implicit models' aforementioned memory efficiency comes at a cost: without good initial guesses, we must spend more iterative updates in order to secure convergence to a good enough estimate of the fixed point in practice, rendering a slower model. However, we argue that the training on many implicit representation tasks, which uses the entire batch of data (e.g., to fit an RGB-color image implicit representation, each pixel of this image is a "sample" of this full batch), can compensate for this issue by providing us very reasonable initial guesses. In particular, training in the full-batch mode allows us to have direct access to *all* of the fixed-points $z^\star_{(t)}$ for all inputs x in the training set immediately after step t. Assuming the model updates between iterations t and $t+1$ are small (which is true in practice, such as via SGD with a small learning rate η), one can safely assume the fixed-points of the updated layer $F(\cdot; x, \theta_{(t+1)})$ do not deviate much from its current estimate $z^\star_{(t)}$. Formally, $F(\cdot; x, \theta_{(t+1)}) \to F(\cdot; x, \theta_{(t)})$ as learning rate $\eta \to 0$, which implies $z^\star_{(t+1)} \to z^\star_{(t)}$. We empirically verify this on these full-batch training settings (see Fig. 3), where a majority of the fixed-point convergences (to a target residual level $\varepsilon = 0.01$) finish in exactly 1 step.

However, we note that this does not imply all training iterations will gain a similar level of convergence boost (e.g., see the first 100 steps in Fig. 3). In particular, at training iteration $t = 0$, we do not have a "previous estimate" at all, which means we still have to solve for $z^\star$ by running the solver for several steps with a neutral initial guess. Thus, we highlight that such fixed-point reuse *amortizes* the cost of later training iterations, and indeed, the majority of the training of the implicit models. Empirically, we found that performing one step of the fixed-point iteration is sufficient for training the iSIREN and iMFN networks; i.e., at training step $t+1$, we simply perform

$$z^\star_{(t+1)} = F(z^\star_{(t)}; x, \theta_{(t)}). \tag{11}$$

Truncated Backward Gradient. As mentioned in §2.2, the backward gradient of implicit models (Eq. (6)) can be alternatively obtained by solving a linear fixed-point system

$$y^\top = y^\top J_F + \frac{\partial \ell}{\partial z^*} \tag{12}$$

so as to avoid expliciting inverting the matrix. However, recent works have suggested that assuming sufficient stability in the DEQ model layer F, we can also approximate $\frac{\partial \ell}{\partial z^\star}\left(I - J_F|_{z^\star}\right)^{-1}$ by an expansion of it as a Neumann series [15]:

$$y^\top = \frac{\partial \ell}{\partial z^\star} \sum_{i=0}^{\infty} \left(J_F\right)^i \tag{13}$$

For example, Fung et al. [12] propose to use the 0-th order approximation and perform a Jacobian-free backward gradient estimation (assuming Lipschitz F); i.e., $y^\top \approx \frac{\partial \ell}{\partial z^\star}$. Although this approximated gradient was empirically shown to be effective in image classification tasks, we found it to substantially hurt the training-time error convergence in implicit representation learning. As a result, we propose to use a truncated backward gradient such that the Neumann series is unrolled for a $T > 0$ steps to better approximate the exact implicit differentiation. Formally, the proposed gradient has the form

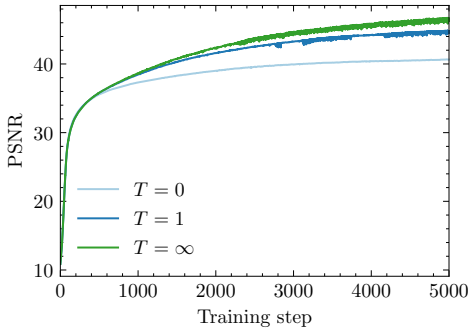

Figure 4: Training PSNR comparison between different levels of gradient approximation. T denotes the approximation order as in Eq. 14, where $T = \infty$ indicates convergence to the solution.

$$\hat{\nabla}_{(\cdot)}\ell = \frac{\partial \ell}{\partial z^*} \sum_{i=0}^{T} \left(J_F\right)^i \frac{\partial F(z^*; x)}{\partial (\cdot)}. \tag{14}$$

In practice, we found $T = 1$ to be already sufficient for implicit representation tasks (see Fig. 4), which requires only one evaluation of the vector-Jacobian product $\frac{\partial \ell}{\partial z^*} J_F$.

Spectral Normalization. We additionally apply spectral normalization on the layers F we use for the implicit models, which ensures that F (for forward passes) and J_F (for backward passes) are contractive mappings, and is able to guarantee unique and stable fixed-points due to the Banach Fixed

Point Theorem [3]. Specifically, we adopt the power-iteration in [21, 4] to scale the spectral norms of W in Eq. (7) and (8) after each training iteration if it become > 1, which incurs little additional computation cost [21]. However, we note that contractivity is not always necessary for fixed-point convergence (especially in the presence of stronger fixed-point solvers, as shown in large-scale DEQ models [1]), and weaker regularizers such as weight normalization [26] may also suffice in practice.

4 Experiments

We evaluate our $(\text{Implicit})^2$ networks on several representation tasks and compare the difference in performance between the implicit and explicit modeling of implicit representations across a variety of configurations. For both explicit networks and implicit networks, we use L to denote the number of layers and D to denote the hidden dimensionality of the features. Unless stated otherwise, our implicit models have exactly one F layer (i.e., $L = 1$), in the exact form we described in §3.1. Our set of experiments is drawn from prior works where competitive implicit representation networks are evaluated on [11, 28].

Overall, our results of learning implicit representation on various domains (including images, videos, audios) suggest that the $(\text{Implicit})^2$ approach offers clear improvements over existing explicit models used for these purposes, where we are able to achieve the same or better level of performance while being up to $3\times$ more memory-efficient and $3\times$ more time-efficient. We provide our implementation[1] and the descriptions of the tasks and datasets can be found in the appendix.

4.1 Image Representation

We first evaluate the difference between explicit networks and $(\text{Implicit})^2$ networks on representing a high-resolution 512×512 grayscale image, which is a commonly used goalpost for evaluating implicit representation models from the scikit-image package [31]. In particular, we fit an implicit representation function $\Phi : (x, y) \to \mathcal{C}$, where $\mathcal{C}$ is the desired color space (and in this case $\mathcal{C} = \mathbb{R}$). We train each model for 5000 iterations (under the same setting) using all pixels in the image (i.e., batch size 262,144), and demonstrate the final peak signal-to-noise ratio (PSNR), memory consumption, and average training step time in Fig. 5. The results convey two interesting facts about using $(\text{Implicit})^2$ networks: Compared to standard 4-layer explicit networks ($4L$-$256D$), even a small implicit network ($1L$-$256D$)

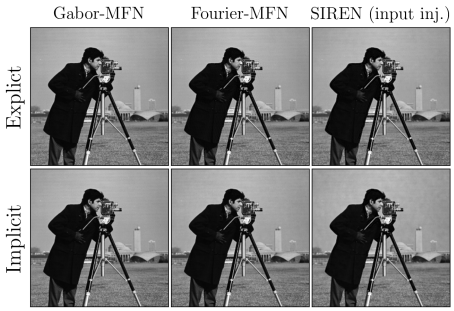

Figure 6: Learned 512×512 grayscale image

[1]Official implementation can be found at `https://github.com/locuslab/ImpSq`

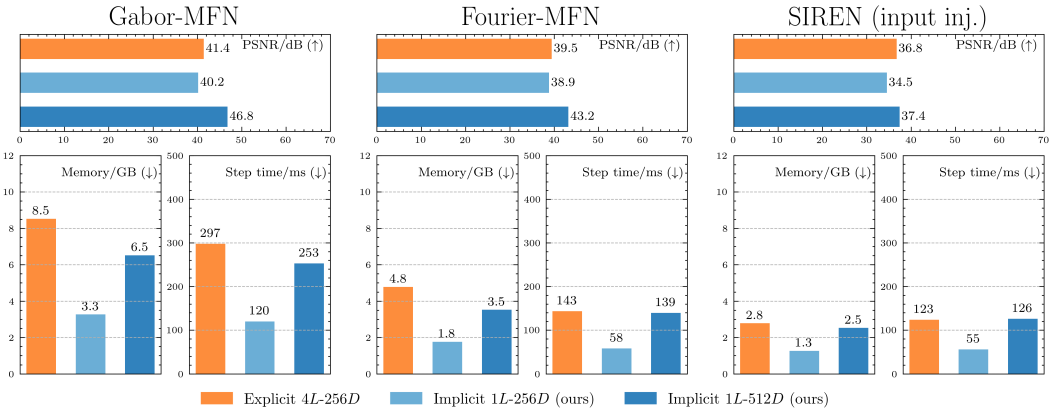

Figure 5: Comparison of step time, memory consumption and PSNR between explicit and implicit models trained on the 512×512 grayscale image.

with approximately 75% fewer parameters achieves a comparable performance, while requiring only roughly 1/3 of the training time and memory footprint. When given a higher parameter budget, an implicit network with a similar model size $(i.e., 1L\text{-}512D)$ outperforms the best explicit network by a large margin.

We also demonstrate the effect of fixed-point reuse and truncated backward gradient using a 1L-512D Fourier MFN on fitting the same 512×512 image, with results shown in Fig. 3 and 4. Specifically, by reusing fixed-points in implicit layers, the solver takes significantly fewer steps to converge for the majority of the training period, making the forward evaluation of implicit layers drastically more efficient. Meanwhile, Fig. 4 shows that the 0th-order gradient approximation in [12] (i.e. $T = 0$), albeit efficient, could substantially hurt model performance. In contrast, our proposed approximation (i.e. $T = 1$) greatly reduces the performance gap with only one additional vector-Jacobian product evaluation, thus achieving a better balance between model efficiency and effectiveness.

4.2 Image Generalization

In a number of applications [20, 22, 5, 8], implicit representations are used to infer unobserved parts of data. In order to demonstrate the ability for $(\text{Implicit})^2$ networks to generalize well, we train the network on only 25% of the pixels from each image in the *Natural* and *Text* dataset, following [11], and evaluate PSNR on an unobserved 25% portion of the image. The average PSNR over all 16 high-resolution images in *Natural* and *Text* is reported in Table 1, and a visualization of the improvement by $(\text{Implicit})^2$ models is shown in Fig. 7.

It can be seen that $(\text{Implicit})^2$ networks improve substantially over the explicit baselines, where the best-performing implicit model outperforms the explicit counterpart by > 0.7 in PSNR using the same architecture. Visually, $(\text{Implicit})^2$ networks produce a sharper and shape-consistent representation compared with the best performing explicit network. We observe similar improvements in training speed and memory consumption as in the image representation task (about $3\times$). For completeness,

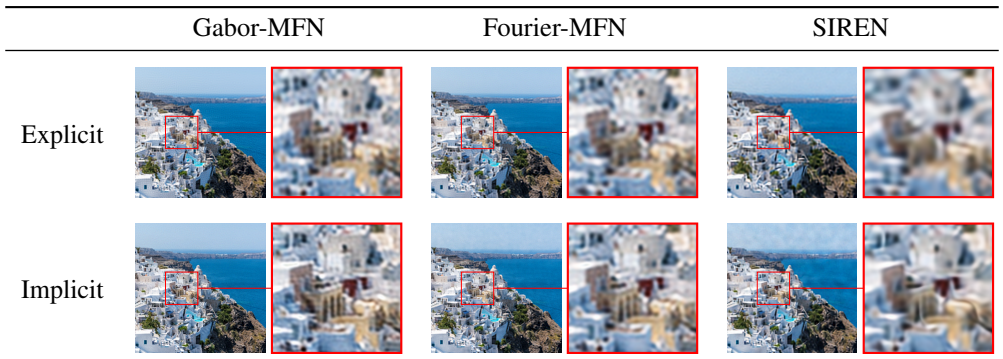

Figure 7: Samples of best performing explicit/implicit models learned on *Natural*

	Natural			*Text*		
	$1L\text{-}256D$	$1L\text{-}512D$	$4L\text{-}256D$	$1L\text{-}256D$	$1L\text{-}512D$	$4L\text{-}256D$
Fourier-MFN	23.27 ± 3.18	23.30 ± 3.05	24.57 ± 3.35	24.64 ± 2.11	24.84 ± 2.10	26.67 ± 2.06
Fourier-iMFN (ours)	24.88 ± .44	**25.19 ± 3.64**	24.52 ± 3.33	26.90 ± 2.14	**27.19 ± 1.83**	26.48 ± 2.04
Gabor-MFN	24.16 ± 3.35	24.68 ± 3.46	24.65 ± 3.38	27.19 ± 2.18	27.74 ± 2.13	27.57 ± 2.10
Gabor-iMFN (ours)	24.91 ± 3.41	**25.42 ± 3.76**	24.53 ± 3.33	27.53 ± 2.18	**28.07 ± 2.00**	27.40 ± 2.10
SIREN (input inj.)	22.88 ± 3.0	24.52 ± 3.28	24.10 ± 3.34	24.54 ± 2.19	25.69 ± 2.18	26.21 ± 2.19
iSIREN (ours)	24.28 ± 3.37	**24.92 ± 3.58**	24.05 ± 3.39	26.06 ± 2.18	**26.81 ± 2.09**	26.31 ± 2.20

Table 1: PSNR (in dB) for all models on image generalization. The reported mean ± std is taken over the individual PSNR of the 16 images.

we also evaluate an (Implicit)^2 network where F also consists of a stack of 4 layers (rather than 1), and observe that the implicit modeling of an already deep structure does not yield much improvement.

4.3 Audio Representation

We further show that the proposed (Implicit)^2 network outperforms explicit networks on representing very-high-frequency audio signals. Following [28], we train the models to fit a 7-second music piece. We observe that in all cases, the implicit modeling significantly outperforms the respective explicit models.

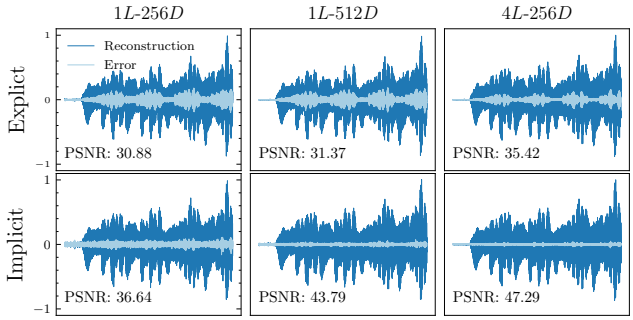

Figure 8: Audio signals represented using Fourier-MFNs. Lower error magnitudes (in light blue) are better.

In Fig. 8, we show the reconstruction results and the respective errors when using Fourier-MFN to fit the audio signal, with more results available in the appendix. Although an (Implicit)^2 model only defines one layer, it is able to outperform explicit models in terms of PSNR by more than 23% (43.79 vs. 35.42), while still retaining the same level of memory and speed improvements as before. We also find that, in this case, having a deeper architecture in an implicit layer (implicit $4L$-$256D$) leads to improved performance.

4.4 Video Representation

We may add an extra dimension to the input of implicit representation function and try to learn a model for video sequences $\Phi : (t, x, y) \to \mathcal{C}$, where t is the space of time. We aim to represent a 300-frame 512×512 video using each model. The PSNR results are shown in Table 2.

	$1L$-$1024D$	$1L$-$2048D$	$4L$-$1024D$
Fourier-MFN	24.97 ± 1.08	26.9 ± 0.97	27.64 ± 0.90
Fourier-iMFN (ours)	25.85 ± 1.00	27.7 ± 0.90	$\mathbf{28.03} \pm \mathbf{0.95}$
Gabor-MFN	26.15 ± 1.02	28.18 ± 0.78	$\mathbf{29.64} \pm \mathbf{0.81}$
Gabor-iMFN (ours)	26.45 ± 0.98	28.79 ± 0.77	29.20 ± 1.04
SIREN (input inj.)	25.12 ± 0.96	26.04 ± 0.99	26.52 ± 0.86
iSIREN (ours)	26.03 ± 0.92	27.08 ± 0.93	$\mathbf{27.12} \pm \mathbf{0.89}$

Table 2: PSNR for the video representation task. The reported mean $\pm$ std is taken over all frames of the video.

Unlike in the image representation setting, we observe that, due to the relatively few parameters compared to the size of the video data (the smallest model only has as many parameters as 1.3% of all pixel values), model sizes have a more significant impact on the overall performance. But still, in most configurations, we found (Implicit)^2 models lead to a consistently non-trivial improvement in performance when compared to their explicit counterparts, while doing so with equivalent or less memory budget.

4.5 3D Geometry Representation

To further demonstrate the benefit of (Implicit)^2 networks over their explicit counterparts, we choose three formulations of Fourier-MFN with similar parameter count (i.e. explicit $1L$-$512D$, explicit $4L$-$256D$, and implicit $1L$-$512D$) and fitted them on several 3D object meshes with the point occupancy prediction objective similar to [19] - given input coordinate $c = (x, y, z)$, the model $\Phi : \mathbb{R}^3 \to \mathbb{R}$ is trained to predict a binary label indicating whether the point corresponding to the coordinate is located inside the target object (0 if the point is outside the object, and 1 if inside). The visualized normal maps of the learned representations on one of the objects are shown in Fig. 9. We further evaluated the prediction IoU over test points densely sampled near the mesh and present them in the same figure. Experimental details and results on additional 3D models are available in the appendix.

The results show that, with a similar model size, the one-layer network modeled with the (Implicit)^2 formulation is able to capture more details than an explicit model with an identical ($1L$-$512D$) or a deeper ($4L$-$256D$) structure, and at the same time having a training time and memory advantage similar to the ones in Fig. 5. Such an example demonstrates the superior parameter and memory efficiency of (Implicit)^2 networks.

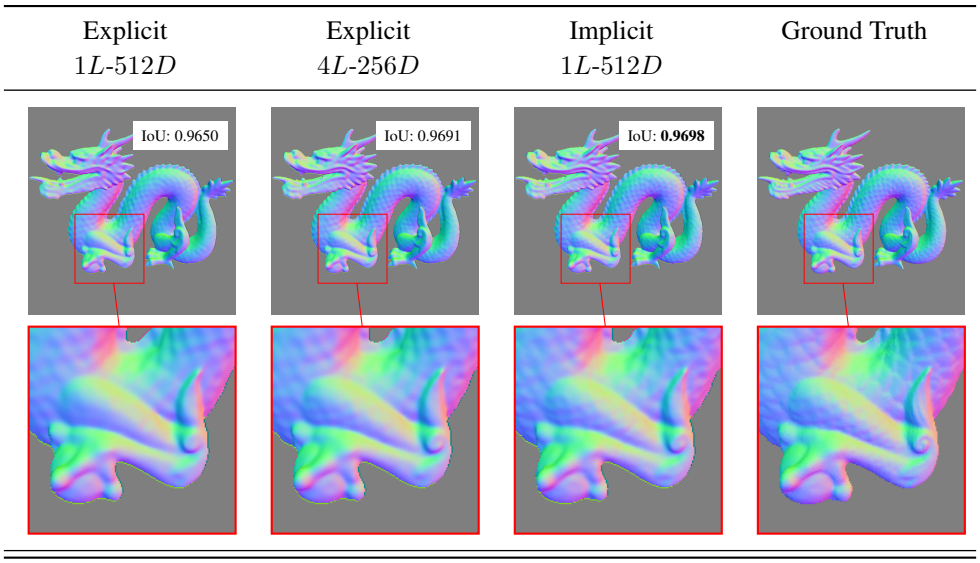

Figure 9: Normal maps and IoUs of fitted *dragon* object using Fourier-MFNs in Occupancy Network [19]. The $(\text{Implicit})^2$ approach performs better than its deep explicit counterpart.

We note that in both the video and 3D geometry tasks, the signal is either of large dimension or defined continuously, where we may not fit the entire data into a single batch. Therefore, the implicit models usually entail more than one step to reach convergence due to the challenge in fixed-point reuse with mini-batch training (see a more detailed analysis in appendix C). Nevertheless, our proposed method is still able to boost the training efficiency by leveraging truncated gradient, while improving upon its explicit counterparts in most cases.

5 Limitations and Directions for Future Work

Our $(\text{Implicit})^2$ approach can be significantly more time-efficient and memory-friendly than explicit networks at training time; however, at inference time, the implicit models must resort to the normal fixed-point solving. Furthermore, accelerated training in the forward pass by fixed point reuse (see §3.2) in practice can be bottlenecked by storage or hardware I/O bandwidth constraints; i.e., it works best when we can fit all or a large portion of the fixed point into memory for quick retrieval. Although such assumption can be easily satisfied for implicit representation learning in many cases, for extremely large data spaces (e.g., in the continuous domain, like Park et al. [23] and Mildenhall et al. [20], where samples are drawn continuously in $\mathbb{R}^3$; or very long video frames like in 4.4), the fixed point reuse may not always work as well and could require more solver steps.

Smarter caching strategies for the fixed points may be developed in the future, where we consider only a subset of the input space for caching throughout the training. For example, one may consider a voxel-grid-based sparse storage for the fixed points, similar to the ones used in [16], such that fixed points of any batch of inputs may be approximately obtained from interpolation of the sparse storage. We leave this direction for future work.

6 Conclusion

In this paper, we propose $(\text{Implicit})^2$ networks, an application of implicit layers to implicit representation tasks. We show that these two concepts of "implicitness" complement each other well, allowing us to take advantage of their properties (e.g., fixed point reuse) to produce a significantly more efficient training routine and usually more parameter-efficient models for implicit representation learning. We demonstrate through our set of experiments that the *implicit modeling* of *implicit representations* may in many cases be conveniently used as a drop-in replacement for existing state-of-the-art explicit models like SIREN and MFN to further improve the performance and reduce the memory/computation budget required to train these tasks.

7 Acknowledgement

We thank Swaminathan Gurumurthy for reviewing the paper and for helpful discussions.

8 Funding Disclosure

Shaojie Bai was sponsored by a grant from the Bosch Center for Artificial Intelligence.

References

[1] S. Bai, J. Z. Kolter, and V. Koltun. Deep equilibrium models. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2019.

[2] S. Bai, V. Koltun, and J. Z. Kolter. Multiscale deep equilibrium models. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2020.

[3] S. Banach. Sur les opérations dans les ensembles abstraits et leur application aux équations intégrales. *Fund. math*, 3(1):133–181, 1922.

[4] J. Behrmann, W. Grathwohl, R. T. Chen, D. Duvenaud, and J.-H. Jacobsen. Invertible residual networks. In *International Conference on Machine Learning*, pages 573–582. PMLR, 2019.

[5] M. Bemana, K. Myszkowski, H.-P. Seidel, and T. Ritschel. X-fields: Implicit neural view-, light- and time-image interpolation. *ACM Transactions on Graphics (Proc. SIGGRAPH Asia 2020)*, 39(6), 2020. doi: 10.1145/3414685.3417827.

[6] C. G. Broyden. A class of methods for solving nonlinear simultaneous equations. *Mathematics of Computation*, 1965.

[7] T. Q. Chen, Y. Rubanova, J. Bettencourt, and D. K. Duvenaud. Neural ordinary differential equations. In *Advances in Neural Information Processing Systems (NIPS)*, 2018.

[8] Y. Chen, S. Liu, and X. Wang. Learning continuous image representation with local implicit image function. *arXiv preprint arXiv:2012.09161*, 2020.

[9] E. Dupont, A. Doucet, and Y. W. Teh. Augmented neural ODEs. In *Advances in Neural Information Processing Systems (NIPS)*, 2019.

[10] D. Duvenaud, J. Z. Kolter, and M. Johnson. Deep implicit layers tutorial - neural ODEs, deep equilibirum models, and beyond. *Neural Information Processing Systems Tutorial*, 2020.

[11] R. Fathony, A. K. Sahu, D. Willmott, and J. Z. Kolter. Multiplicative filter networks. In *International Conference on Learning Representations*, 2021. URL `https://openreview.net/forum?id=OmtmcPkkhT`.

[12] S. W. Fung, H. Heaton, Q. Li, D. McKenzie, S. Osher, and W. Yin. Fixed point networks: Implicit depth models with Jacobian-free backprop. *arXiv:2103.12803*, 2021.

[13] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[14] S. G. Krantz and H. R. Parks. *The implicit function theorem: History, theory, and applications*. Springer, 2012.

[15] R. Liao, Y. Xiong, E. Fetaya, L. Zhang, K. Yoon, X. Pitkow, R. Urtasun, and R. Zemel. Reviving and improving recurrent back-propagation. In *International Conference on Machine Learning (ICML)*, 2018.

[16] L. Liu, J. Gu, K. Z. Lin, T.-S. Chua, and C. Theobalt. Neural sparse voxel fields. *NeurIPS*, 2020.

[17] Z. Liu, P. Luo, X. Wang, and X. Tang. Deep learning face attributes in the wild. In *Proceedings of International Conference on Computer Vision (ICCV)*, December 2015.

[18] C. Lu, J. Chen, C. Li, Q. Wang, and J. Zhu. Implicit normalizing flows. In *International Conference on Learning Representations (ICLR)*, 2021.

[19] L. Mescheder, M. Oechsle, M. Niemeyer, S. Nowozin, and A. Geiger. Occupancy networks: Learning 3d reconstruction in function space. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4460–4470, 2019.

[20] B. Mildenhall, P. P. Srinivasan, M. Tancik, J. T. Barron, R. Ramamoorthi, and R. Ng. Nerf: Representing scenes as neural radiance fields for view synthesis. In *European Conference on Computer Vision*, pages 405–421. Springer, 2020.

[21] T. Miyato, T. Kataoka, M. Koyama, and Y. Yoshida. Spectral normalization for generative adversarial networks. In *International Conference on Learning Representations*, 2018. URL `https://openreview.net/forum?id=B1QRgziT-`.

[22] M. Niemeyer, L. Mescheder, M. Oechsle, and A. Geiger. Differentiable volumetric rendering: Learning implicit 3d representations without 3d supervision. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 3504–3515, 2020.

[23] J. J. Park, P. Florence, J. Straub, R. Newcombe, and S. Lovegrove. Deepsdf: Learning continuous signed distance functions for shape representation. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

[24] D. W. Peaceman and H. H. Rachford, Jr. The numerical solution of parabolic and elliptic differential equations. *Journal of the Society for industrial and Applied Mathematics*, 3(1): 28–41, 1955.

[25] M. Revay, R. Wang, and I. R. Manchester. Lipschitz bounded equilibrium networks. *arXiv:2010.01732*, 2020.

[26] T. Salimans and D. P. Kingma. Weight normalization: A simple reparameterization to accelerate training of deep neural networks. In *Advances in Neural Information Processing Systems (NIPS)*, 2016.

[27] V. Sitzmann, M. Zollhöfer, and G. Wetzstein. Scene representation networks: Continuous 3d-structure-aware neural scene representations. In *Advances in Neural Information Processing Systems*, 2019.

[28] V. Sitzmann, J. N. Martel, A. W. Bergman, D. B. Lindell, and G. Wetzstein. Implicit neural representations with periodic activation functions. In *Proc. NeurIPS*, 2020.

[29] M. Tancik, P. P. Srinivasan, B. Mildenhall, S. Fridovich-Keil, N. Raghavan, U. Singhal, R. Ramamoorthi, J. T. Barron, and R. Ng. Fourier features let networks learn high frequency functions in low dimensional domains. *NeurIPS*, 2020.

[30] A. Tewari, O. Fried, J. Thies, V. Sitzmann, S. Lombardi, K. Sunkavalli, R. Martin-Brualla, T. Simon, J. Saragih, M. Nießner, et al. State of the art on neural rendering. In *Computer Graphics Forum*, volume 39, pages 701–727. Wiley Online Library, 2020.

[31] S. van der Walt, J. L. Schönberger, J. Nunez-Iglesias, F. Boulogne, J. D. Warner, N. Yager, E. Gouillart, T. Yu, and the scikit-image contributors. scikit-image: image processing in Python. *PeerJ*, 2:e453, 6 2014. ISSN 2167-8359. doi: 10.7717/peerj.453. URL `https://doi.org/10.7717/peerj.453`.

[32] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems (NIPS)*, 2017.

[33] E. Winston and J. Z. Kolter. Monotone operator equilibrium networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2020.

A Proof for the Stability of iMFN and iSIREN

In this section, we show that, once we restrict W in Eq. (7) and (8) in the main text to be contractive via spectral normalization, the function F in both iMFN and iSIREN will also be contractive, which therefore guarantees the existence and uniqueness of a fixed point [3]. For the remaining part of this proof, we use $\|\cdot\|_2$ on a matrix to denote its spectral norm and $\sigma(\cdot)$ to denote the set of its eigenvalues.

Theorem 1 (Contractivity of iMFN)**.** *Let $\|W\|_2 < 1$ and $\|g(x;\theta)\|_\infty \leq 1$ for all input x and parameter θ. The function F for the iMFN, e.g.*

$$F(z;x) = (W(z + g(x;\theta_1)) + b) \circ g(x;\theta_2) \tag{15}$$

is contractive on $\mathcal{Z}$.

Proof. Let $G_{x,\theta}$ be a diagonal matrix with $g(x;\theta)$ as the main diagonal, e.g.

$$G_{x,\theta} = \begin{bmatrix} g_1(x;\theta) & 0 & \cdots & 0 \\ 0 & g_2(x;\theta) & & \\ \vdots & & \ddots & \vdots \\ 0 & \cdots & & g_d(x;\theta) \end{bmatrix} \tag{16}$$

where $g_i(x;\theta)$ is a scalar-valued function representing the i-th element of $g(x;\theta)$. It is easy to see that $\|G_{x,\theta}\|_2 \leq 1$ since $\max_{\lambda\in\sigma(G_{x,\theta}^\top G_{x,\theta})} |\lambda| = \max_{i\in[d]} g_i^2(x;\theta) \leq 1$. Therefore, for any $z_1, z_2 \in \mathcal{Z}$, we have

$$\begin{aligned}
\frac{\|F(z_1;x) - F(z_2;x)\|_2}{\|z_1 - z_2\|_2} &= \frac{\|Wz_1 \circ g(x;\theta_2) - Wz_2 \circ g(x;\theta_2)\|_2}{\|z_1 - z_2\|_2} && (17)\\
&= \frac{\|G_{x,\theta_2} W(z_1 - z_2)\|_2}{\|z_1 - z_2\|_2} && (18)\\
&\leq \frac{\|G_{x,\theta_2}\|_2 \|W\|_2 \|z_1 - z_2\|_2}{\|z_1 - z_2\|_2} && (19)\\
&= \|G_{x,\theta_2}\|_2 \|W\|_2 && (20)\\
&< 1 && (21)
\end{aligned}$$

which shows that F is contractive on $\mathcal{Z}$. Additionally, the Jacobian matrix J_F can be directly obtained by $J_F = G_{x,\theta} W$, which is also contractive since $\|G_{x,\theta} W\|_2 \leq \|G_{x,\theta}\|_2 \|W\|_2 < 1$ □

Theorem 2 (Contractivity of iSIREN)**.** *Let $\|W\|_2 < 1$. The function F for the iSIREN, e.g.*

$$F(z;x) = \sin(W(z + \sin(Vx)) + Ux + b) \tag{22}$$

is contractive and has a contractive Jacobian J_F.

Proof. Since the sine function is 1-Lipschitz, we have

$$\begin{aligned}
\frac{\|F(z_1;x) - F(z_2;x)\|_2}{\|z_1 - z_2\|_2} &= \frac{\|\sin(W(z_1 + \sin(Vx)) + Ux + b) - \sin(W(z_2 + \sin(Vx)) + Ux + b)\|_2}{\|z_1 - z_2\|_2} && (23)\\
&\leq \frac{\|W(z_1 + \sin(Vx)) + Ux + b - W(z_2 + \sin(Vx)) + Ux + b\|_2}{\|z_1 - z_2\|_2} && (24)\\
&= \frac{\|W(z_1 - z_2)\|_2}{\|z_1 - z_2\|_2} && (25)\\
&\leq \|W\|_2 && (26)\\
&< 1 && (27)
\end{aligned}$$

which shows the contractivity of F.

We now show the contractivity of the Jacobian J_F. Let $h = W(z + \sin(Vx)) + Ux + b$ and D be a diagonal matrix with $\cos(h)$ as the main diagonal. There is $\|D\|_2 = \max_{\lambda \in \sigma(D^\top D)} |\lambda| = \max_{i \in [d]} \cos^2(h_i) \leq 1$. Therefore, the Jacobian J_F is clearly contractive since it can be written explicitly as $J_F = \frac{\partial \sin(h)}{\partial h} \frac{\partial h}{\partial z} = DW$, where $\|J_F\|_2 = \|DW\|_2 \leq \|D\|_2 \|W\|_2 < 1$. □

B Proof for §3.1

We shall proof the claim in §3.1 that the outputs of iMFN are linear combinations of the non-linear filter kernels. The proof largely follows [11], where we show that for a filter function that satisfies the *multiplicative sum* property (as defined below), every application of F yields a linear combination of filter functions.

Definition 1 (Fathony et al. [11])**.** *A filter function $g(x;\theta)$ satisfies the **multiplicative sum property** if for any input x and any two parameters configurations θ_1, θ_2, there exists a finite set of parameter configurations $\{\overline{\theta}_i\}_{i \in [N]}$ and scalars $\{\beta_i\}_{i \in [n]}$ such that*

$$g(x;\theta_1) \circ g(x;\theta_2) = \sum_{i=0}^{N} \beta_i g(x;\overline{\theta}_i) \tag{28}$$

By A.1 and A.2 of [11], both g_{Fourier} and g_{Gabor} of the original MFN satisfies the multiplicative sum property. We then have the following corollary

Corollary 1. *Define $z \in \mathbb{R}^d$ such that $z_k = \sum_{i=1}^{N} \beta_i^{(k)} g(x;\theta_i^{(k)})$ with parameters $\{\beta_i^{(k)}\}_{i \in [N]}$, $\{\theta_i^{(k)}\}_{i \in [N]}$ and $g(x;\theta)$ satisfies the multiplicative sum property. The output of the function*

$$F(z;x) = (W(z + g(x;\theta')) + b) \circ g(x;\theta'') \tag{29}$$

satisfies $F_j(z;x) = \sum_{i=1}^{\overline{N}} \overline{\beta}_i^{(j)} g(x;\overline{\theta}_i^{(j)})$ with some other parameters $\{\overline{\beta}_i^{(j)}\}_{i \in [\overline{N}]}$, $\{\overline{\theta}_i^{(j)}\}_{i \in [\overline{N}]}$.

Proof.

$$\begin{aligned}
F_j(z;x) &= \left(\sum_{k=1}^{d} W_{jk} \left(z_k + g(x;\theta'_k) \right) + b_j \right) g(x;\theta''_j) && (30)\\
&= \left(\sum_{k=1}^{d} W_{jk} \sum_{i=1}^{N} \beta_i^{(k)} g(x;\theta_i^{(k)}) + \sum_{k=1}^{d} W_{jk} g(x;\theta'_k) + b_j \right) g(x;\theta''_j) && (31)\\
&= \sum_{k=1}^{d} \sum_{i=1}^{N} W_{jk} \beta_i^{(k)} g(x;\theta_i^{(k)}) g(x;\theta''_j) + \sum_{k=1}^{d} W_{jk} g(x;\theta'_k) g(x;\theta''_j) + b_j g(x;\theta''_j) && (32)\\
&= \sum_{i=1}^{\overline{N}} \overline{\beta}_i g(x;\overline{\theta}_i) && (33)
\end{aligned}$$

□

Corollary 1 states that, if every element of z is a linear combination of filter functions, applying F once also yields a vector consists of linear combinations of filter functions. Further, assuming the contractivity conditions in §A, the fixed point of F in Eq. (29) exists and is unique. Thus, starting from $z^{[0]} = \mathbf{0}^d$, the unique fixed point $z^\star$ may be defined recursively through the fixed-point iteration with infinite steps, namely

$$z^\star = z^{[\infty]}, \qquad \text{where } z^{[t+1]} = F(z^{[t]};x),\ z^{[0]} = \mathbf{0}^d \tag{34}$$

By recursively applying Corollary 1, we have $\forall t > 0, \forall k \in [d]$, $z_k^{[t]}$ is a linear combination of filter functions, which proves the same property for $z^\star$. Since the outputs of iMFN is a linear transformation of $z^\star$, each of its elements remains a linear combination of filter functions, which finishes the proof.

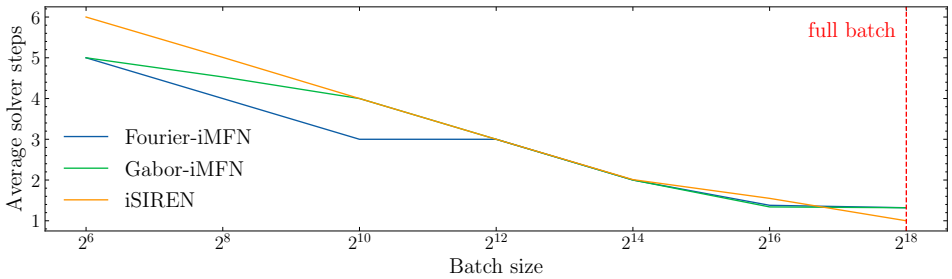

Figure 10: Average fixed-point solver steps *versus* mini-batch sizes. The average steps are taken over the last 1,000 training iterations of each model. From the figure, it can be seen that the required solver steps is inversely related to the batch size. However, only when the batch size is sufficiently large ($> 2^{16}$) can the majority of the fixed-points converge in 1 step (average solver steps < 1.5)

.

C (Implicit)^2 Model and SGD

The explicit models we compared to in this work (except ones for video and 3D occupancy representations) are trained with full batch gradient descent. Given the success of stochastic gradient descent (SGD) in deep learning, it is natural to wonder whether training the explicit models in mini-batches would yield better training performance or generalization. Empirically, we find the answer to be *no*, where we evaluated different mini-batch sizes {64, 256, 1024, 4096} in the image generalization task and, given the same compute budget, find the resulting models to still be inferior to (Implicit)^2 networks both in terms of training and testing error.

Another natural question to ask is whether (Implicit)^2 models, especially the fixed-point reuse component, can be applied in SGD training as well, since SGD is often more feasible in general machine learning tasks. To explore this, we maintain a fixed-point cache for each training sample (i.e. a coordinate) in the image fitting task, and experiment with different batch sizes at which the fixed-points are computed by running the ordinary fixed-point solving routine. We cached the fixed-points at every training iteration and reuse them whenever a sample is revisited. As the training error stabilizes, the number of steps the solver takes to converge from the cached fixed points is reported in Fig. 10.

It is clear from the table that replacing the fixed point iteration with a single forward iteration (which is required to match the efficiency of the explicit models) is only feasible when the batch consists of a significant portion of the training data. Therefore, the proposed replacement of the fixed-point solver, in general, can only work with data of relative low dimensions, such as images coordinates (which is extensively studied in this paper). Nevertheless, it can also be observed that the fixed-point cost saving is proportional to the mini-batch size, which may motivate fixed-point reuse in DEQ trained on other tasks as a general strategy to save computation. A potentially more interesting future direction, as covered in §5, is to leverage some spatial-temporal priors of the data (e.g. spatially close points in 3D space or consecutive frames in a video stream) to further reduce the storage and updates needed to maintain the cache.

D Training Specifications

In this section, we provide detailed training specifications and hyperparameter choices used to produce our main results. Unless specified otherwise, we apply spectral normalization as in [4] on all W and used an Adam optimizer with ($\beta_1 = 0.9, \beta_2 = 0.999, \epsilon =$ 1e-8) for training.

D.1 Image Representation

Data. The 512×512 grayscale image is imported from the scikit-image package [31] of version 0.16.2 via skimage.data.camera(). We notice a recent change to this image in the latest scikit-image package due to copyright issues, and therefore call attention to the readers to use the correct data.

Initialization. We apply a scaling of 256 to all input layers and choose $\alpha = 3.0$ for the initialization of the Gabor filters

Training Details. We use all (262,144) pixels in each batch and train the models for $5,000$ steps with a constant learning rate 1e-3.

Hardware. The models are trained and evaluated using a 11GB NVIDIA GTX 1080TI GPU.

Additional Note. Figure 3 and 4 in the main text are generated using a 1L-512D Fourier-iMFN with the same data and hyperparameters, except that we control whether the fixed points are reused and the number of Jacobian evaluations T during the backward pass, respectively.

D.2 Image Generalization

Data. The *Natural* and *Text* dataset we use in this experiment was originally created by [29] and consists of 16 natural or word overlay images of resolution 512×512. The data is made available alongside the code.

Initialization. We apply a scaling of 128 to all input layers and choose $\alpha = 3.0$ for the initialization of the Gabor filters

Training Details. Following prior works like Tancik et al. [29], we train the networks on 25% of the data and evaluate on a disjoint 25% set (a visualization is shown in Fig. 11). For every model, we train on each image for $2,000$ steps with learning rate $\eta =$ 1e-3.

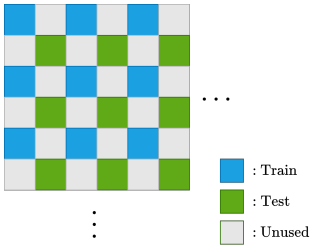

Figure 11: Train/Test split in the image generalization task

Hardware. The models are trained and evaluated using a 11GB NVIDIA GTX 1080TI GPU.

D.3 Audio Representation

Data. Following [28], the audio signal is retrieved from the first 7-seconds of Bach's Cello Suite No. 1: Prelude and is also made available in the submitted code.

Initialization. To account for the high-frequency nature of the audio signals, we applied a scaling of 25,000 to the input linear layers. We empirically find that implicit models without spectral norm yields better results while remains stable throughout training. Therefore, in the audio experiments, we apply weight normalization on W instead of spectral normalization.

Training Details. We train the models using the entire audio piece in each batch (of size $308,980$) for $3,000$ steps with a constant learning rate 1e-3.

Hardware. The models are trained and evaluated using a 11GB NVIDIA RTX 2080TI GPU.

D.4 Video Representation

Data. The full video clip used in the video representation task is publicly available at

https://www.pexels.com/video/the-full-facial-features-of-a-pet-cat-3040808

We choose a 512×512 center crop of this video for training and evaluation, following [28, 11]. The data is also included as part of our submitted code.

Initialization. We apply the same input scaling to both explicit and implicit models with the same base architectures according to Table 3. The individual scaling factors were chosen to optimize the performance of the explicit networks. Additionally, we choose $\alpha = 6.0$ in the 4-layer models and $\alpha = 3.0$ in the 1-layer models for the initialization of the Gabor filters. We apply weight normalization instead of spectral normalization on all W.

Training Details. We train each model for $10,000$ steps with a constant learning rate 1e-3 and a batch size of 50,000. The batch size is chosen such that the largest explicit model (i.e. Gabor-MFN-$4L$-$1024D$) is able to fit in the GPU memory.

	$1L$-$1024D$	$1L$-$2048D$	$4L$-$1024D$
Gabor-MFN	128	256	256
Fourier-MFN	128	256	256
SIREN (input inj.)	128	128	256

Table 3: Scaling factors for explicit/implicit models

Hardware. The models are trained and evaluated using a 11GB NVIDIA RTX 2080TI GPU.

D.5 3D Model Occupancy Representation

Data. The 3D objects (i.e. *dragon*, *buddha*, *armadillo*, *lucy*) used in this experiment are retrieved from the Stanford 3D Scanning Repository.

Initialization. We apply a scaling of 256 to all input layers.

Training/Evaluation Details. Our training and evaluation procedure largely follows Tancik et al. [29]. The 3D object meshes are scaled and transformed to fit in side the unit cube $[0, 1]^3$. We use the point-in-mesh algorithm in the Trimesh library to efficiently obtain the ground truth point occupancy label for the input coordinates. Each model is trained on the binary cross-entropy for $10,000$ steps with a *constant* learning rate 5e-4. At inference, the depth maps are generated based on the first ray intersection to the isosurface $\Phi(x, y, z) = 0.5$ from the camera, and the normal maps (as shown in Fig. 9) are computed using derivatives of the depth maps. The test points used to compute IoU are sampled from *i.i.d.* Gaussians, each with a mean at uniformly random point on the mesh and a standard deviation of 0.01.

Hardware. The models are trained and evaluated using a 11GB NVIDIA RTX 2080TI GPU.

E Additional Visualizations & Experiments

E.1 Image fitting on CelebA

We evaluated both explicit and implicit models on fitting each of the first 100 images in the CelebA dataset [17]. The images were center-cropped and rescaled to 128×128. We used a learning rate of 3e-3 and trained each network for 3,000 steps. The results are presented in Table 4. This experiment shows that the $(\text{Implicit})^2$ networks still outperform explicit counterparts on data of larger scales, further demonstrating their effectiveness.

	$1L$-$128D$	$1L$-$256D$	$4L$-$128D$
Fourier-MFN	35.48 ± 4.01	36.41 ± 4.30	44.27 ± 2.75
Fourier-iMFN	38.03 ± 2.78	$\mathbf{46.70 \pm 2.78}$	44.72 ± 2.45
Gabor-MFN	39.08 ± 3.52	45.85 ± 3.70	42.55 ± 2.89
Gabor-iMFN	39.39 ± 2.83	$\mathbf{47.72 \pm 2.02}$	41.96 ± 3.17
SIREN	34.41 ± 3.70	36.97 ± 3.98	38.78 ± 4.06
iSIREN	36.75 ± 3.62	$\mathbf{40.90 \pm 3.08}$	39.40 ± 4.07

Table 4: PSNR for image fitting on CelebA

E.2 Gabor-MFN and SIREN for Audio Representation

As promised in §4.3 of the main text, we show the results of the remaining two models in the audio representation tasks in Fig 12 and 13. We observe that implicit Gabor-MFN outperforms its

explicit counterparts, while iSIREN performs slightly worse. We hypothesize the marginally inferior performance of iSIREN is likely due to the absence of spectral normalization (as mentioned in Sec D.3), which makes implicit models less stable toward the end of training. Empirically, iSIREN suffers the most from this instability which thus causes the relatively weaker performance. However, even with such instability, the best implicit models still perform competitively to the best explicit models, while have a better memory/training time profile due to implicit modeling with fixed-point reuse and truncated backward pass.

E.3 Additional Visualizations for Image Generalization

We show additional outputs of learned implicit representation models in the image generalization task (Fig. 14 and Fig. 15). The models we compare (implicit $1L$-$512D$ and explicit $4L$-$256D$) have a similar parameter count, while the implicit network in general requires less memory and training time due to the fixed point reuse and truncated backward gradient. For all types of networks (Gabor-MFN, Fourier-MFN, and SIREN), the $(\text{Implicit})^2$ formulation consistently outperform the explicit counterparts in visual quality, while also enjoys more computational efficiency.

E.4 Additional Visualizations for 3D Object Occupancy

We show the visualized normal maps for 3 additional 3D objects from Stanford 3D Scanning Repository, i.e. *buddha*, *armadillo*, and *lucy*, in Fig. 16. The results show that the shallow implicit network ($1L$-$512D$) yields reconstructions with details on par or better than the deep explicit alternative ($4L$-$256D$) and significantly outperforms the shallow explicit network ($1L$-$512D$).

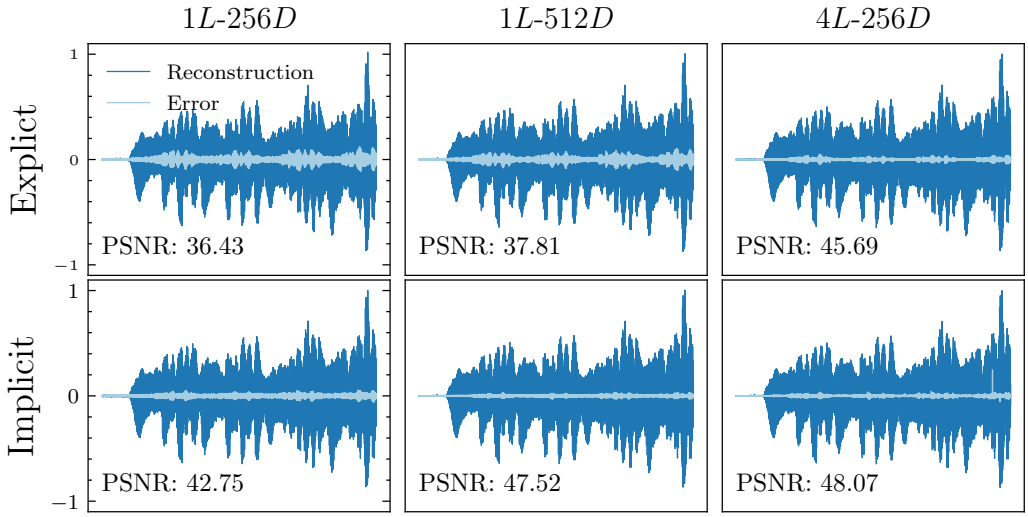

Figure 12: Audio representation task using explicit/implicit Gabor-MFN

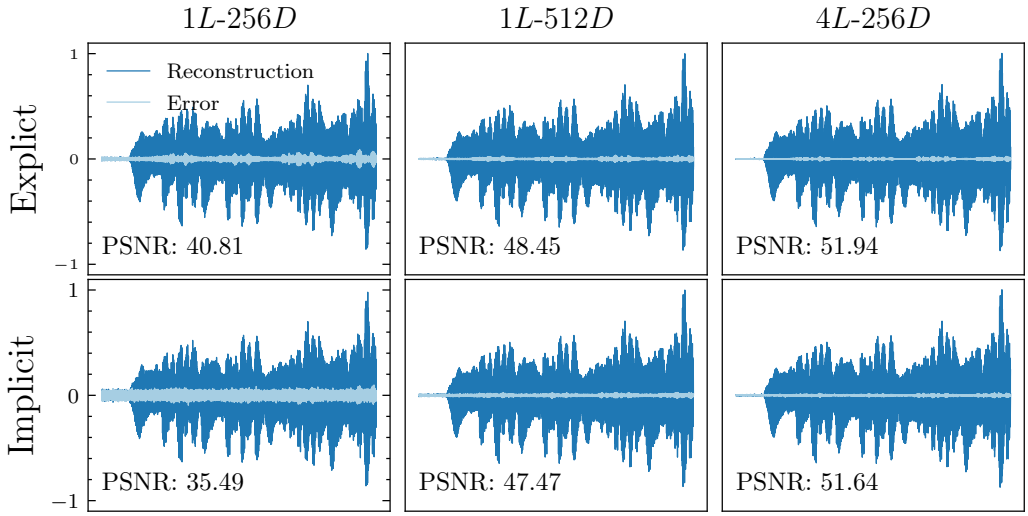

Figure 13: Audio representation task using SIREN (input inj.) or iSIREN

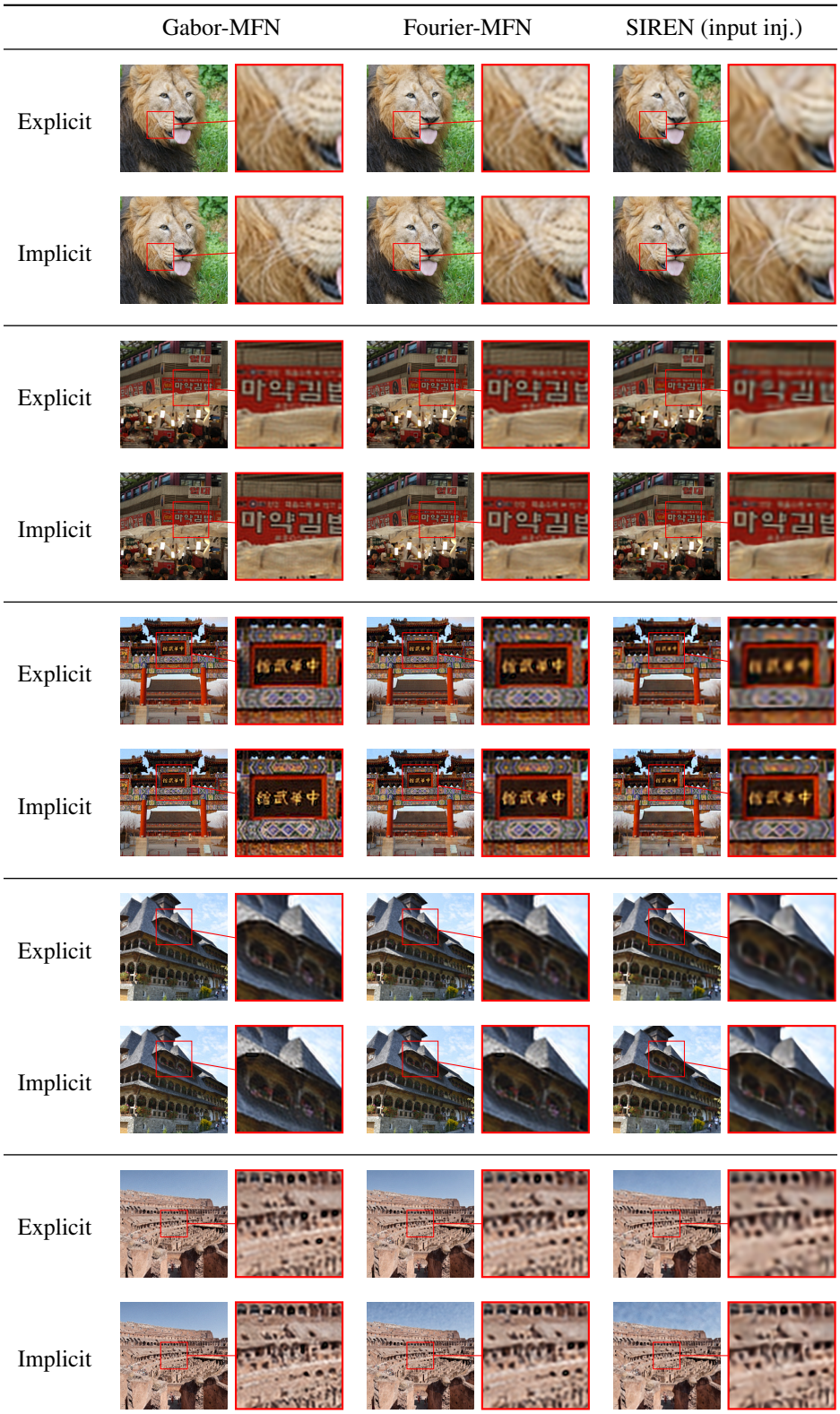

Figure 14: Implicit $1L$-$512D$ *vs.* Explicit $4L$-$256D$ on *Natural*

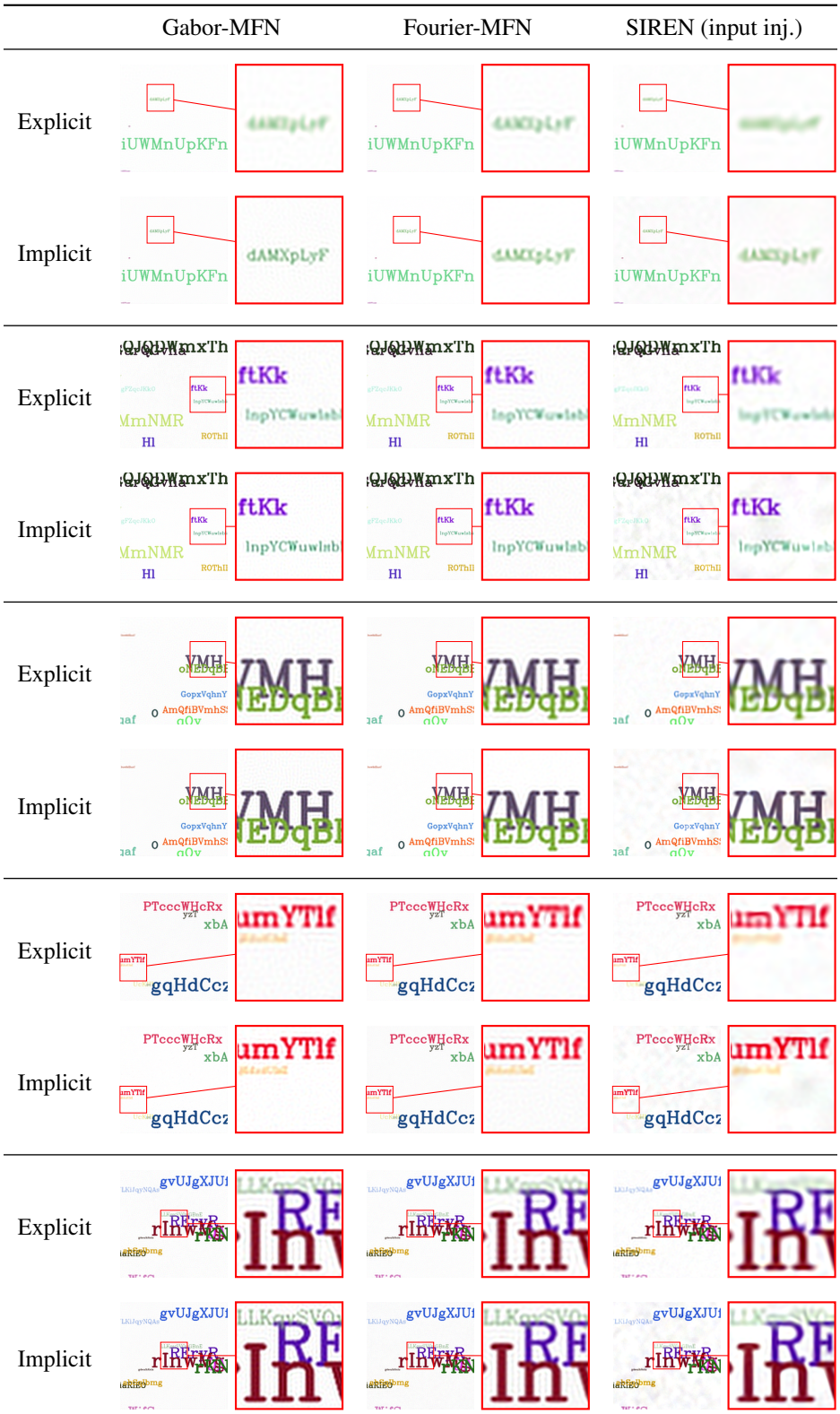

Figure 15: Implicit $1L$-$512D$ *vs.* Explicit $4L$-$256D$ on *Text*

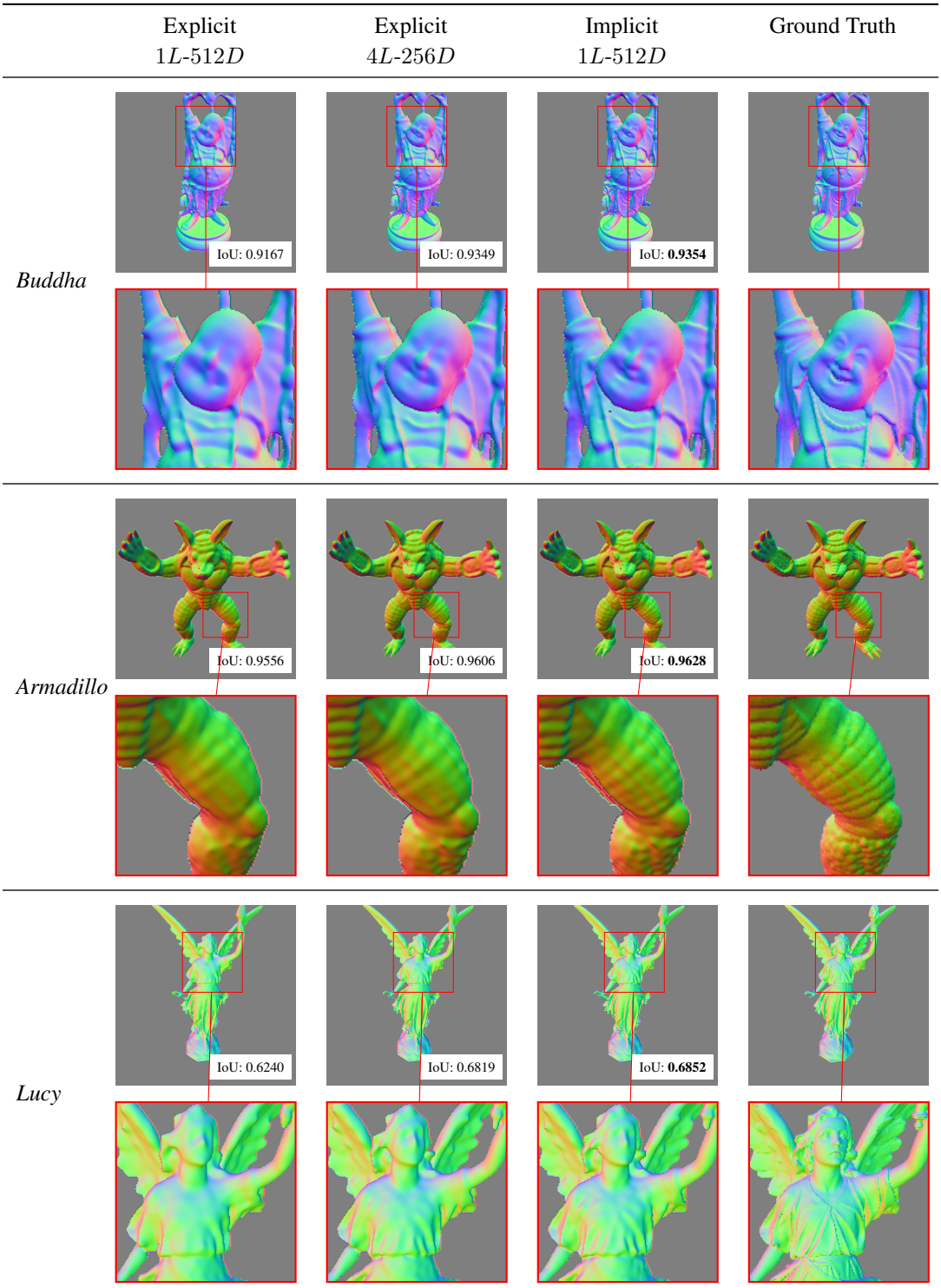

Figure 16: Additional results on fitting objects from Stanford 3D Scanning Repository

F NeurIPS2021 Paper Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes] We provide a detailed description in §5.
   (c) Did you discuss any potential negative societal impacts of your work? [No] We did not spot any particular links to negative societal impacts for our work.
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [N/A]
   (b) Did you include complete proofs of all theoretical results? [Yes] We include proofs in the appendix.
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes] We include the code and experimental details in the supplement
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] We provide a detailed specification in the supplement
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes] We report the variance of image generalization results on 16 different images
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] We provide details in the supplement
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes]
   (b) Did you mention the license of the assets? [No] We are not aware of particular licenses they are released under.
   (c) Did you include any new assets either in the supplemental material or as a URL? [No]
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [No] The data we used are all publicly available in previous works.
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [No] We are not aware of any personally identifiable information or offensive content in the data.
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]