Joint Inference and Input Optimization in DEQs

Paper by: Gurumurthy, Bai, Manchester, and Kolter

Slides by Paul Vicol

Optimization Over Inputs

- Several tasks in machine learning require *optimization over inputs*
  - Finding a *latent representation* that decodes to an image matching a target:

$$\arg\min_{z} ||x - G_\theta(z)||_2^2$$

  - *Adversarial training*

$$\max_{||\delta|| \leq \epsilon} \ell(g(x+\delta), y)$$

  - A range of *inverse problems*, etc.

Gurumurthy et al., “Joint Inference and Input Optimization in DEQs” NeurIPS 2021.

Optimization Over Inputs

- Several tasks in machine learning require *optimization over inputs*
  - Finding a *latent representation* that decodes to an image matching a target:

$$\arg\min_{z} ||x - G_\theta(z)||_2^2$$

  - *Adversarial training*

$$\max_{||\delta|| \leq \epsilon} \ell(g(x+\delta), y)$$

  - A range of *inverse problems*, etc.

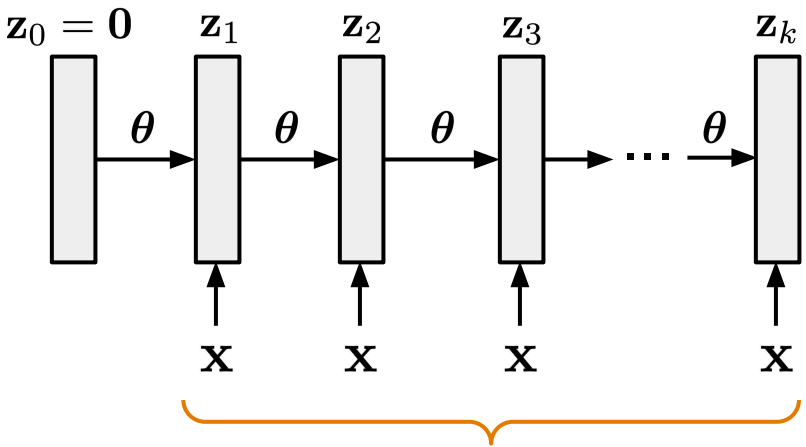
- **Problem:** *Optimizing over inputs is costly*, since each gradient step requires a forward + backward pass through the network

- **This paper** combines the *input optimization with fixpoint iteration of DEQs*
- **Goal:** *Speed up training* of networks that involve such “bilevel” structure

Gurumurthy et al., “Joint Inference and Input Optimization in DEQs” NeurIPS 2021.

DEQ Recap: Weight-Tied, Input-Injected Layers

Weight-Tied, Input-Injected Architecture



The "effective depth" is given by the number of iterations of the *single layer* $f_\theta(\mathbf{z}, \mathbf{x})$

- Such architectures were studied in the 1990s (as "convergent RNNs")

$$\mathbf{z}_{i+1} = f_\theta(\mathbf{z}_i, \mathbf{x})$$

- RNNs are the most well-known examples of such weight-tied models, although typically they have different inputs at each step rather than the same input re-injected
- Recently, *weight-tied models have made a resurgence, e.g., universal transformers*

DEQ Recap: Taking the Number of Layers to Infinity

- If we take the *number of layers to infinity*, then under some conditions we *converge to a fixpoint*

$$\mathbf{z}^\star = f_{\boldsymbol{\theta}}(\mathbf{z}^\star, \mathbf{x})$$

- **Main idea behind deep equilibrium models (DEQs):**
  - Define the output of a layer to be the *solution to the fixpoint equation* $\mathbf{z}^\star = f_{\boldsymbol{\theta}}(\mathbf{z}^\star, \mathbf{x})$

- **Forward pass:** *arbitrary fixpoint solver* to find z_star
- **Backward pass:** *implicit differentiation*
  - Can be done in different ways, either with another fixpoint solve, or with Neumann series

Optimization Over Inputs

- We aim to solve: $\arg\min_{\mathbf{x}} \ell(\mathbf{z}^*_\theta(\mathbf{x}), y)$

Gurumurthy et al., “Joint Inference and Input Optimization in DEQs” NeurIPS 2021.

Optimization Over Inputs

- We aim to solve: $\arg\min_{\mathbf{x}} \ell(\mathbf{z}^*_\theta(\mathbf{x}), y)$
- Typically, we optimize x with gradient descent: $\mathbf{x}^+ = \mathbf{x} - \alpha \frac{\partial \ell(\mathbf{z}^*_\theta(\mathbf{x}), y)}{\partial \mathbf{x}}$

Gurumurthy et al., “Joint Inference and Input Optimization in DEQs” NeurIPS 2021.

Optimization Over Inputs

- We aim to solve: $\arg\min_{\mathbf{x}} \ell(\mathbf{z}^*_\theta(\mathbf{x}), y)$
- Typically, we optimize x with gradient descent: $\mathbf{x}^+ = \mathbf{x} - \alpha \frac{\partial \ell(\mathbf{z}^*_\theta(\mathbf{x}), y)}{\partial \mathbf{x}}$
- Using the chain rule, $\frac{\partial \ell(\mathbf{z}^*_\theta(\mathbf{x}), y)}{\partial \mathbf{x}} = \frac{\partial \ell(\mathbf{z}^*_\theta(\mathbf{x}), y)}{\partial \mathbf{z}^*_\theta(\mathbf{x})} \boxed{\frac{\partial \mathbf{z}^*_\theta(\mathbf{x})}{\partial \mathbf{x}}}$

Gurumurthy et al., “Joint Inference and Input Optimization in DEQs” NeurIPS 2021.

Optimization Over Inputs

- We aim to solve: $\arg\min_{\mathbf{x}} \ell(\mathbf{z}^*_\theta(\mathbf{x}), y)$
- Typically, we optimize x with gradient descent: $\mathbf{x}^+ = \mathbf{x} - \alpha \frac{\partial \ell(\mathbf{z}^*_\theta(\mathbf{x}), y)}{\partial \mathbf{x}}$
- Using the chain rule, $\frac{\partial \ell(\mathbf{z}^*_\theta(\mathbf{x}), y)}{\partial \mathbf{x}} = \frac{\partial \ell(\mathbf{z}^*_\theta(\mathbf{x}), y)}{\partial \mathbf{z}^*_\theta(\mathbf{x})} \frac{\partial \mathbf{z}^*_\theta(\mathbf{x})}{\partial \mathbf{x}}$

- Using implicit differentiation: $\mathbf{z}^*_\theta(\mathbf{x}) = f_\theta(\mathbf{z}^*_\theta(\mathbf{x}), \mathbf{x})$

Gurumurthy et al., “Joint Inference and Input Optimization in DEQs” NeurIPS 2021.

Optimization Over Inputs

- We aim to solve: $\arg\min_{\mathbf{x}} \ell(\mathbf{z}^*_\theta(\mathbf{x}), y)$
- Typically, we optimize x with gradient descent: $\mathbf{x}^+ = \mathbf{x} - \alpha \frac{\partial \ell(\mathbf{z}^*_\theta(\mathbf{x}), y)}{\partial \mathbf{x}}$
- Using the chain rule, $\frac{\partial \ell(\mathbf{z}^*_\theta(\mathbf{x}), y)}{\partial \mathbf{x}} = \frac{\partial \ell(\mathbf{z}^*_\theta(\mathbf{x}), y)}{\partial \mathbf{z}^*_\theta(\mathbf{x})} \boxed{\frac{\partial \mathbf{z}^*_\theta(\mathbf{x})}{\partial \mathbf{x}}}$
- Using implicit differentiation:

$$\mathbf{z}^*_\theta(\mathbf{x}) = f_\theta(\mathbf{z}^*_\theta(\mathbf{x}), \mathbf{x})$$

$$\frac{d\mathbf{z}^*_\theta(\mathbf{x})}{d\mathbf{x}} = \frac{df_\theta(\mathbf{z}^*_\theta(\mathbf{x}), \mathbf{x})}{d\mathbf{x}}$$

Gurumurthy et al., “Joint Inference and Input Optimization in DEQs” NeurIPS 2021.

Optimization Over Inputs

- We aim to solve: $\arg\min_{\mathbf{x}} \ell(\mathbf{z}^*_\theta(\mathbf{x}), y)$
- Typically, we optimize x with gradient descent: $\mathbf{x}^+ = \mathbf{x} - \alpha \frac{\partial \ell(\mathbf{z}^*_\theta(\mathbf{x}), y)}{\partial \mathbf{x}}$
- Using the chain rule, $\frac{\partial \ell(\mathbf{z}^*_\theta(\mathbf{x}), y)}{\partial \mathbf{x}} = \frac{\partial \ell(\mathbf{z}^*_\theta(\mathbf{x}), y)}{\partial \mathbf{z}^*_\theta(\mathbf{x})} \boxed{\frac{\partial \mathbf{z}^*_\theta(\mathbf{x})}{\partial \mathbf{x}}}$
- Using implicit differentiation:

$$\mathbf{z}^*_\theta(\mathbf{x}) = f_\theta(\mathbf{z}^*_\theta(\mathbf{x}), \mathbf{x})$$

$$\frac{d\mathbf{z}^*_\theta(\mathbf{x})}{d\mathbf{x}} = \frac{df_\theta(\mathbf{z}^*_\theta(\mathbf{x}), \mathbf{x})}{d\mathbf{x}}$$

$$\frac{d\mathbf{z}^*_\theta(\mathbf{x})}{d\mathbf{x}} = \frac{\partial f_\theta(\mathbf{z}^*_\theta(\mathbf{x}), \mathbf{x})}{\partial \mathbf{x}} + \frac{\partial f_\theta(\mathbf{z}^*_\theta(\mathbf{x}), \mathbf{x})}{\partial \mathbf{z}^*_\theta(\mathbf{x})} \frac{d\mathbf{z}^*_\theta(\mathbf{x})}{d\mathbf{x}}$$

Gurumurthy et al., “Joint Inference and Input Optimization in DEQs” NeurIPS 2021.

Optimization Over Inputs

- We aim to solve: $\arg\min_{\mathbf{x}} \ell(\mathbf{z}^*_\theta(\mathbf{x}), y)$
- Typically, we optimize x with gradient descent: $\mathbf{x}^+ = \mathbf{x} - \alpha \frac{\partial \ell(\mathbf{z}^*_\theta(\mathbf{x}), y)}{\partial \mathbf{x}}$
- Using the chain rule, $\frac{\partial \ell(\mathbf{z}^*_\theta(\mathbf{x}), y)}{\partial \mathbf{x}} = \frac{\partial \ell(\mathbf{z}^*_\theta(\mathbf{x}), y)}{\partial \mathbf{z}^*_\theta(\mathbf{x})} \boxed{\frac{\partial \mathbf{z}^*_\theta(\mathbf{x})}{\partial \mathbf{x}}}$
- Using implicit differentiation:

$$\mathbf{z}^*_\theta(\mathbf{x}) = f_\theta(\mathbf{z}^*_\theta(\mathbf{x}), \mathbf{x})$$

$$\frac{d\mathbf{z}^*_\theta(\mathbf{x})}{d\mathbf{x}} = \frac{df_\theta(\mathbf{z}^*_\theta(\mathbf{x}), \mathbf{x})}{d\mathbf{x}}$$

$$\frac{d\mathbf{z}^*_\theta(\mathbf{x})}{d\mathbf{x}} = \frac{\partial f_\theta(\mathbf{z}^*_\theta(\mathbf{x}), \mathbf{x})}{\partial \mathbf{x}} + \frac{\partial f_\theta(\mathbf{z}^*_\theta(\mathbf{x}), \mathbf{x})}{\partial \mathbf{z}^*_\theta(\mathbf{x})} \frac{d\mathbf{z}^*_\theta(\mathbf{x})}{d\mathbf{x}}$$

$$\left(I - \frac{\partial f_\theta(\mathbf{z}^*_\theta(\mathbf{x}), \mathbf{x})}{\partial \mathbf{z}^*_\theta(\mathbf{x})}\right) \frac{d\mathbf{z}^*_\theta(\mathbf{x})}{d\mathbf{x}} = \frac{\partial f_\theta(\mathbf{z}^*_\theta(\mathbf{x}), \mathbf{x})}{\partial \mathbf{x}}$$

Gurumurthy et al., “Joint Inference and Input Optimization in DEQs” NeurIPS 2021.

Optimization Over Inputs

- We aim to solve: $\arg\min_{\mathbf{x}} \ell(\mathbf{z}^*_\theta(\mathbf{x}), y)$
- Typically, we optimize x with gradient descent: $\mathbf{x}^+ = \mathbf{x} - \alpha \frac{\partial \ell(\mathbf{z}^*_\theta(\mathbf{x}), y)}{\partial \mathbf{x}}$
- Using the chain rule, $\frac{\partial \ell(\mathbf{z}^*_\theta(\mathbf{x}), y)}{\partial \mathbf{x}} = \frac{\partial \ell(\mathbf{z}^*_\theta(\mathbf{x}), y)}{\partial \mathbf{z}^*_\theta(\mathbf{x})} \boxed{\frac{\partial \mathbf{z}^*_\theta(\mathbf{x})}{\partial \mathbf{x}}}$
- Using implicit differentiation:

$$\mathbf{z}^*_\theta(\mathbf{x}) = f_\theta(\mathbf{z}^*_\theta(\mathbf{x}), \mathbf{x})$$

$$\frac{d\mathbf{z}^*_\theta(\mathbf{x})}{d\mathbf{x}} = \frac{df_\theta(\mathbf{z}^*_\theta(\mathbf{x}), \mathbf{x})}{d\mathbf{x}}$$

$$\frac{d\mathbf{z}^*_\theta(\mathbf{x})}{d\mathbf{x}} = \frac{\partial f_\theta(\mathbf{z}^*_\theta(\mathbf{x}), \mathbf{x})}{\partial \mathbf{x}} + \frac{\partial f_\theta(\mathbf{z}^*_\theta(\mathbf{x}), \mathbf{x})}{\partial \mathbf{z}^*_\theta(\mathbf{x})} \frac{d\mathbf{z}^*_\theta(\mathbf{x})}{d\mathbf{x}}$$

$$\left(I - \frac{\partial f_\theta(\mathbf{z}^*_\theta(\mathbf{x}), \mathbf{x})}{\partial \mathbf{z}^*_\theta(\mathbf{x})}\right) \frac{d\mathbf{z}^*_\theta(\mathbf{x})}{d\mathbf{x}} = \frac{\partial f_\theta(\mathbf{z}^*_\theta(\mathbf{x}), \mathbf{x})}{\partial \mathbf{x}}$$

$$\boxed{\frac{d\mathbf{z}^*_\theta(\mathbf{x})}{d\mathbf{x}}} = \left(I - \frac{\partial f_\theta(\mathbf{z}^*_\theta(\mathbf{x}), \mathbf{x})}{\partial \mathbf{z}^*_\theta(\mathbf{x})}\right)^{-1} \frac{\partial f_\theta(\mathbf{z}^*_\theta(\mathbf{x}), \mathbf{x})}{\partial \mathbf{x}}$$

Gurumurthy et al., “Joint Inference and Input Optimization in DEQs” NeurIPS 2021.

Optimization Over Inputs

- They combine the update for the input x with the fixpoint update for z as:

$$\begin{bmatrix} z^+ \\ x^+ \end{bmatrix} := \begin{bmatrix} f_\theta(z,x) \\ x - \alpha \left(\frac{\partial f_\theta(z,x)}{\partial x}\right)^\top \underbrace{\left(I - \frac{\partial f_\theta(z,x)}{\partial z}\right)^{-\top}} \left(\frac{\partial \ell(z,y)}{\partial z}\right)^\top \end{bmatrix}$$

Computing the inverse Jacobian is expensive

Gurumurthy et al., “Joint Inference and Input Optimization in DEQs” NeurIPS 2021.

Optimization Over Inputs

- They combine the update for the input x with the fixpoint update for z as:

$$\begin{bmatrix} z^+ \\ x^+ \end{bmatrix} := \begin{bmatrix} f_\theta(z,x) \\ x - \alpha\left(\frac{\partial f_\theta(z,x)}{\partial x}\right)^\top \underbrace{\left(I - \frac{\partial f_\theta(z,x)}{\partial z}\right)^{-\top}} \left(\frac{\partial \ell(z,y)}{\partial z}\right)^\top \end{bmatrix}$$

Computing the inverse Jacobian is expensive

- Computing the inverse Jacobian is like the standard backward pass of a DEQ, where we have to find the JVP:

$$\mu = \left(I - \frac{\partial f_\theta(\mathbf{z},\mathbf{x})}{\partial \mathbf{z}}\right)^{-1} \frac{\partial \ell(\mathbf{z},y)}{\partial \mathbf{z}}$$

$$\left(I - \frac{\partial f_\theta(\mathbf{z},\mathbf{x})}{\partial \mathbf{z}}\right)\mu = \frac{\partial \ell(\mathbf{z},y)}{\partial \mathbf{z}}$$

$$\boxed{\mu = \frac{\partial f_\theta(\mathbf{z},\mathbf{x})}{\partial \mathbf{z}}\mu + \frac{\partial \ell(\mathbf{z},\mathbf{x})}{\partial \mathbf{z}}} \leftarrow \textit{Another fixpoint iteration}$$

Gurumurthy et al., “Joint Inference and Input Optimization in DEQs” NeurIPS 2021.

Optimization Over Inputs

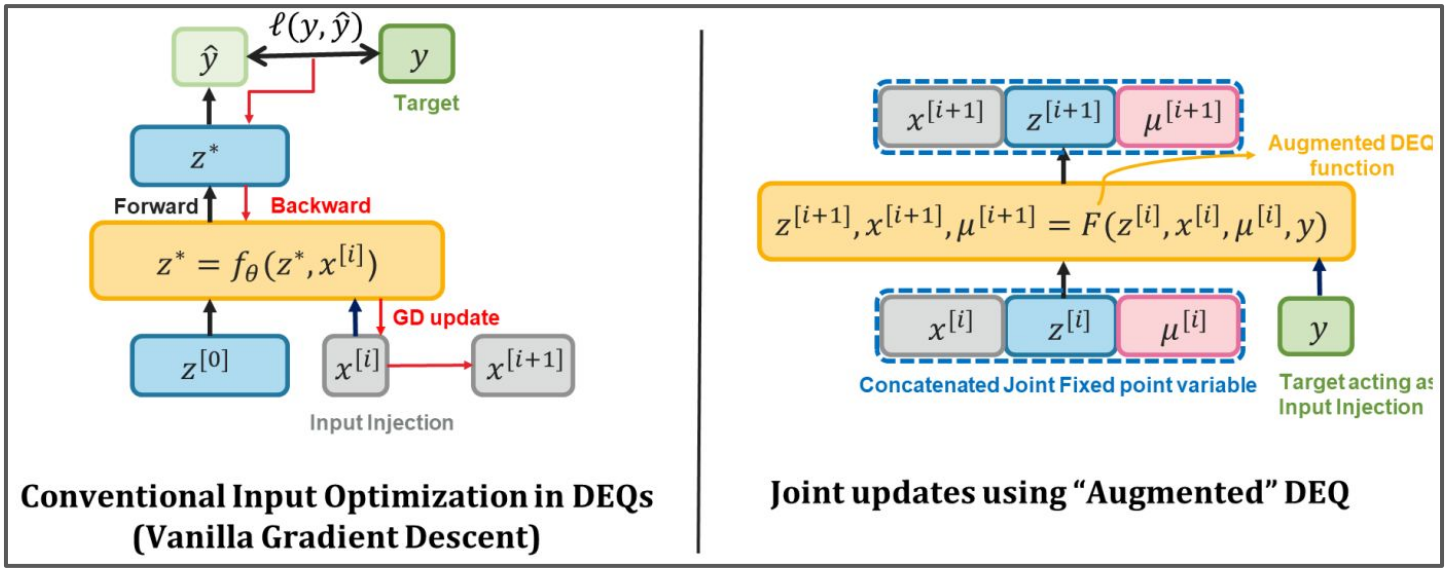
- So, they further propose to combine this fixpoint iteration over mu into the update:

$$\begin{bmatrix} z^+ \\ \mu^+ \\ x^+ \end{bmatrix} := \underbrace{\begin{bmatrix} f(z,x) \\ \left(\frac{\partial f_\theta(z,x)}{\partial z}\right)^\top \mu + \left(\frac{\partial \ell(z,y)}{\partial z}\right)^\top \\ x - \alpha\left(\frac{\partial f_\theta(z,x)}{\partial x}\right)^\top \mu \end{bmatrix}}$$

They interpret this as an “augmented” DEQ network

Gurumurthy et al., “Joint Inference and Input Optimization in DEQs” NeurIPS 2021.

Optimization Over Inputs



Gurumurthy et al., "Joint Inference and Input Optimization in DEQs" NeurIPS 2021.

Experiment Overview

- **Experiments**
  1. Training DEQ-based generative models while optimizing over latent codes
  2. Training models for inverse problems such as denoising or inpainting
  3. Adversarial training of implicit models
  4. Gradient-based meta-learning

- **Claims**
  - Simultaneous input optimization and DEQ training is faster than a naive inner/outer optimization
    - 3.5-9x speedup for generative DEQ networks
    - 3x speedup in adversarial training of DEQs
    - 2.5-3x speedup for meta-learning

Gurumurthy et al., “Joint Inference and Input Optimization in DEQs” NeurIPS 2021.

Generative Modeling

- Learning a *decoder-only generative model* — computes latents by directly minimizing a reconstruction loss.

$$\begin{aligned} &\underset{\theta}{\text{minimize}} \quad \sum_{i=1}^{n} \|y_i - h_\theta(z_i^\star)\|^2 \\ &\text{subject to} \quad x_i^\star, z_i^\star = \underset{x, z: z = f_\theta(z, x)}{\text{argmin}} \quad \|y_i - h_\theta(z)\|^2, \; i = 1, \ldots, n \end{aligned}$$

Figure 2: Samples generated with JIIO on a small MDEQ network.

Model	Generation	Reconstruction
VAE [43]	48.12	39.12
RAE [24]	**40.96**	36.01
MDEQ-VAE	57.15	45.81
MDEQ-VAE (w/ JIIO)	-	42.36
JIIO-MDEQ	46.82	**32.52**

Table 1: Comparison of FID scores attained by standard generative models with our method, which performs joint optimization. We use 40 solver iterations (for the augmented DEQ) to train the JIIO model reported in this table.

Model	time taken (ms)
Adam : 40 iters	7659
JIIO : 40 iters	862
JIIO : 100 iters	2156

Table 3: Time taken to perform JIIO optimization v/s Adam in the generative modeling/inverse problem experiments

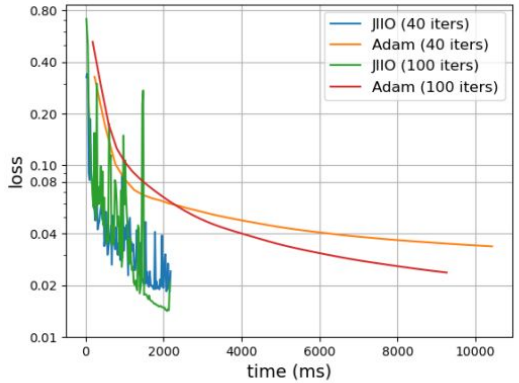


Figure 3: Cost changing with time for Adam v/s JIIO optimization. Tested on models trained with 40 and 100 JIIO iterations respectively

Gurumurthy et al., “Joint Inference and Input Optimization in DEQs” NeurIPS 2021.

Adversarial Training

- JIIO allows for training implicit models that are similarly robust to models trained with PGD, more quickly than PGD
- Train MDEQ on CIFAR10 and MNIST using adv training with epsilon=1.

$$\underset{\theta}{\text{minimize}} \quad \sum_{i=1}^{n} \max_{\|\delta\|_2 \leq \epsilon} \ell(h_\theta(x_i + \delta_i), y_i)$$

$$\begin{aligned} &\underset{\theta}{\text{minimize}} \quad \sum_{i=1}^{n} \ell(h_\theta(z_i^\star), y) \\ &\text{subject to} \quad z_i^\star, \delta_i^\star = \underset{z,\delta: z = f_\theta(z, x+\delta), \|\delta\|_2 \leq \epsilon}{\text{argmin}} -\ell(h_\theta(z), y), \; i = 1, \ldots, n \end{aligned}$$

Datasets	**Train (↓) Test (→)**	Clean	PGD	JIIO
MNIST	Clean	99.45 ± 0.03	80.1 ± 1.87	65.88 ± 4.72
	PGD	99.18 ± 0.03	96.53 ± 0.05	95.74 ± 0.04
	JIIO	99.32 ± 0.09	95.74 ± 0.22	96.63 ± 0.58
CIFAR	Clean	78.47 ± 0.94	2.38 ± 0.41	3.71 ± 4.01
	PGD	54.91 ± 1.01	37.4 ± 0.26	36.17 ± 0.55
	JIIO	55.54 ± 0.82	37.31 ± 0.67	37.77 ± 0.84

Table 5: Comparison of adversarial training approaches on L2 norm perturbations with $\epsilon = 1$. The rows represent the training procedure and the columns represent the testing procedure

Model	**time taken (ms)**
PGD : 20 iters	4360
JIIO : 80 iters	1401

Table 2: Time taken to compute adversarial example of a MDEQ model on MNIST

Gurumurthy et al., “Joint Inference and Input Optimization in DEQs” NeurIPS 2021.

Drawbacks

- Requires way more fixpoint iterations to converge (50-100) vs a standard DEQ (10-20)
  - Uses more memory depending on which fixpoint solver you use (e.g., Anderson or Broyden, etc.)

Gurumurthy et al., “Joint Inference and Input Optimization in DEQs” NeurIPS 2021.

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

Paper by: Zhichun Huang, Shaojie Bai, Zico Kolter

Slides by Paul Vicol

Implitictness

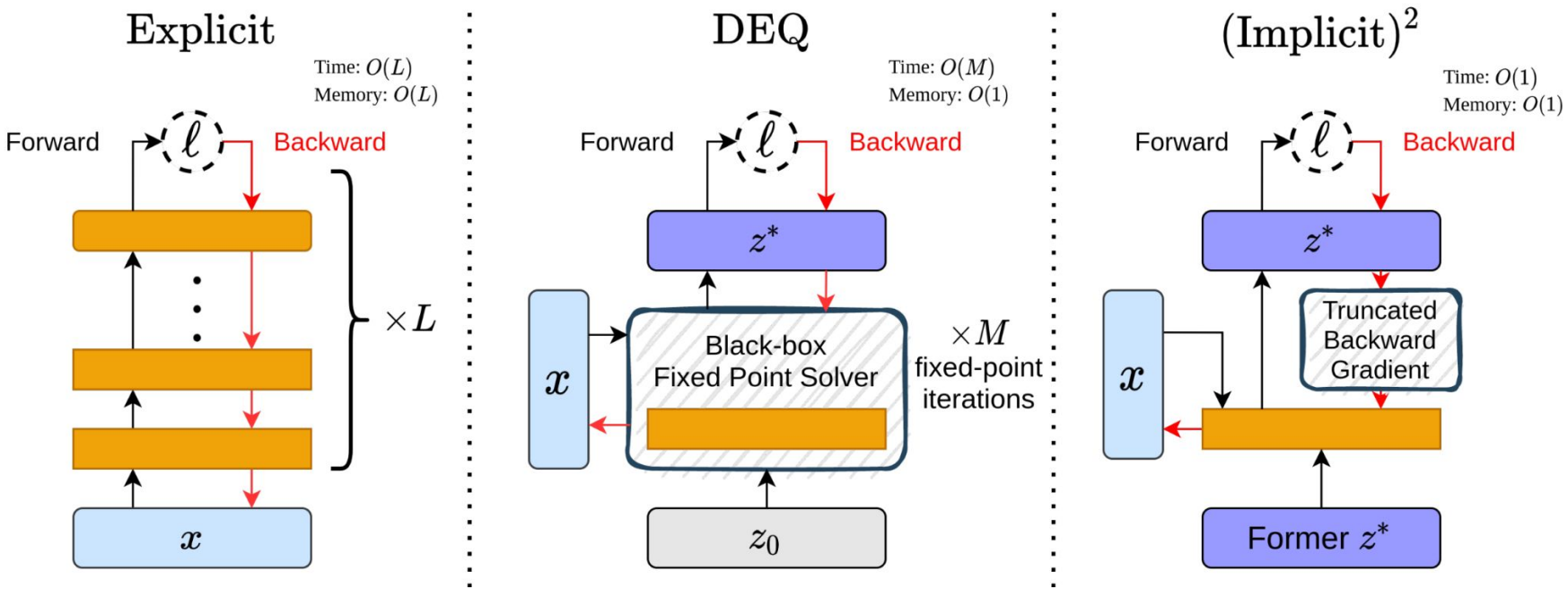
- Two different types of “implicitness”
  - **Implicit representations:** continuous representations of high-frequency data like images, audio, or video using a low-dimensional neural net
  - **Implicit layers:** effectively “infinite-depth” models whose forward pass is computed by solving a fixpoint equation

- **Plot twist:** this paper applies *implicit layers to learn implicit representations*

- They define *simple implicit (single-repeated-layer) variants* of the *SIREN* and *Multiplicative Filter Networks (MFN) models*
  - And they apply their variants (iSIREN and iMFN) to several domains
  - Comparing against explicit versions of these models

Huang et al., “Implicit Layers for Implicit Representations” NeurIPS 2021.

Training Tricks

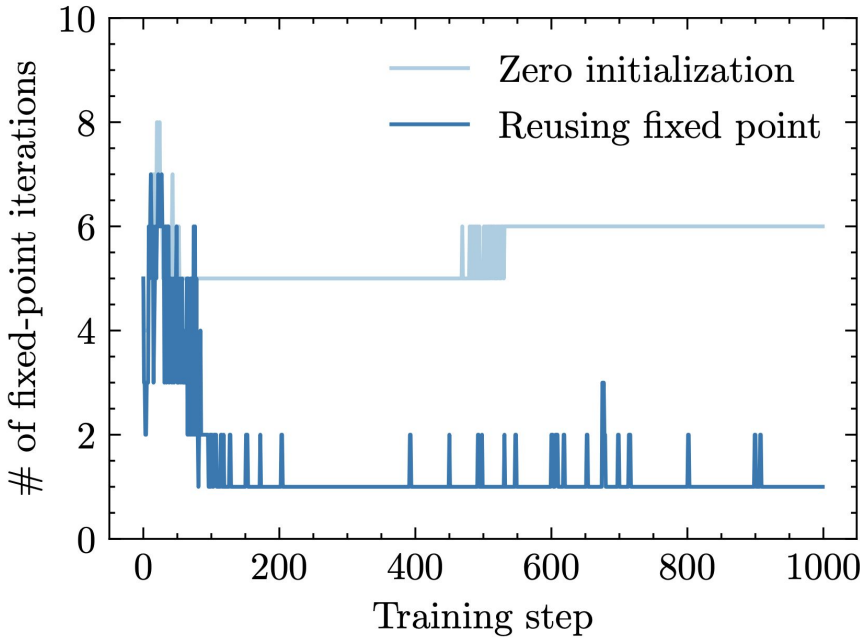


Huang et al., “Implicit Layers for Implicit Representations” NeurIPS 2021.

Amortized Fixpoint Iterations

- Implicit representations are typically trained in *full-batch mode* (e.g., operating on all the pixels in an input image simultaneously, where each pixel position is a separate input)
- The fixpoint z from one forward pass will probably be similar to the fixpoint in the next forward pass
  - Because the parameters don't change much in each step

- **Side note:** amortizing the forward/backward passes may also be useful for time series, or when subsequent inputs are similar to each other
  - Diffusion models, video frames, etc.



Huang et al., "Implicit Layers for Implicit Representations" NeurIPS 2021.

Faster Backward Pass: Truncated Neumann Iterations

- *Truncated Neumann series* iterations to compute the implicit gradient work well
- This was also found by several other papers

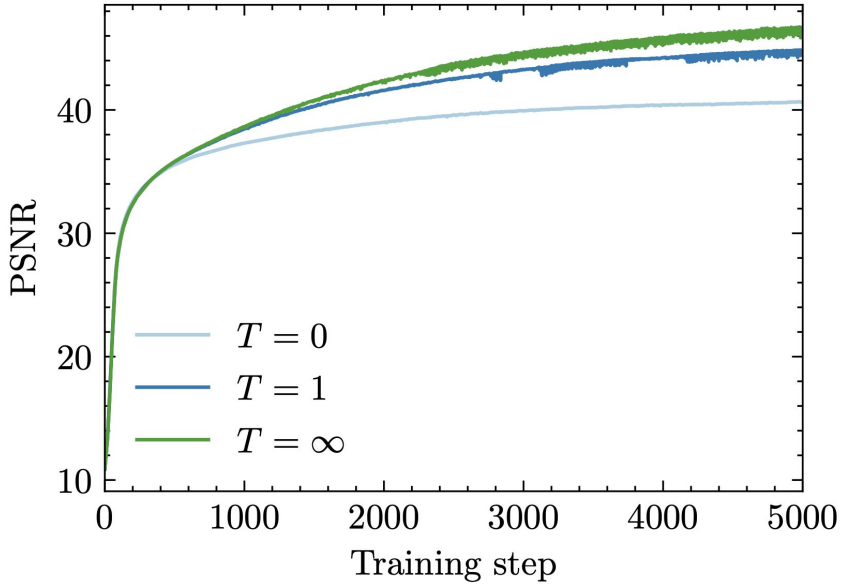
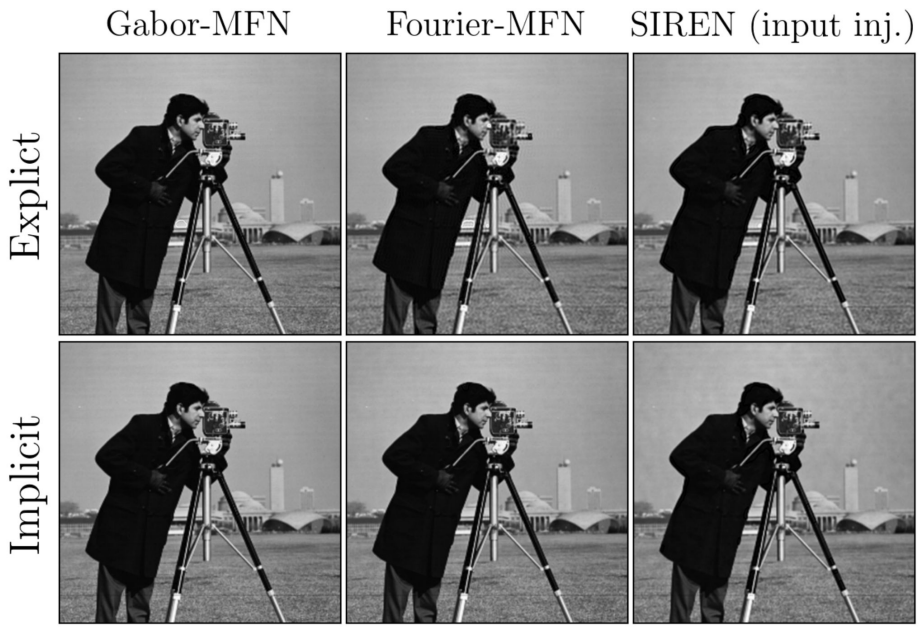


Figure 4: Training PSNR comparison between different levels of gradient approximation. T denotes the approximation order as in Eq. 14, where $T = \infty$ indicates convergence to the solution.

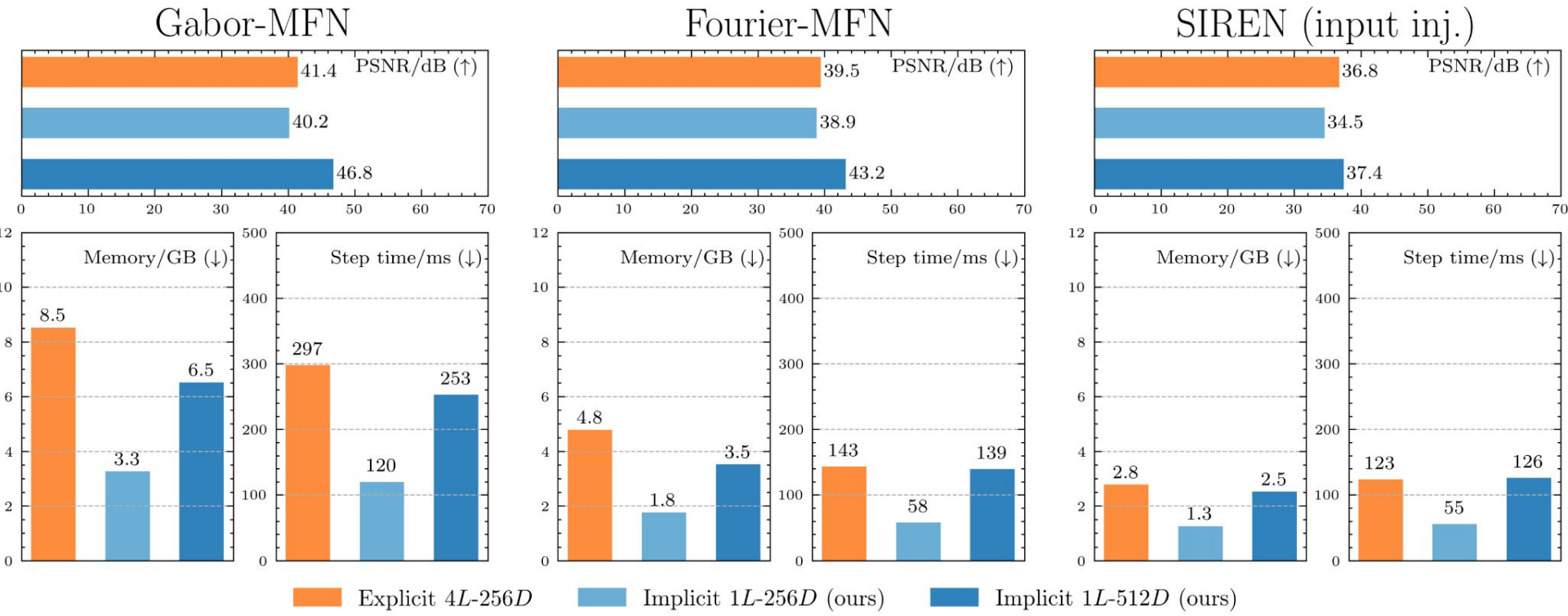
Huang et al., “Implicit Layers for Implicit Representations” NeurIPS 2021.

Image Representation



Huang et al., “Implicit Layers for Implicit Representations” NeurIPS 2021.

Image Representation



Huang et al., “Implicit Layers for Implicit Representations” NeurIPS 2021.

Image Generalization

- Train on only 25% of the pixels from each image and evaluate PSNR on an unobserved 25% portion of the image

	Gabor-MFN	Fourier-MFN	SIREN
Explicit			
Implicit			

Huang et al., “Implicit Layers for Implicit Representations” NeurIPS 2021.

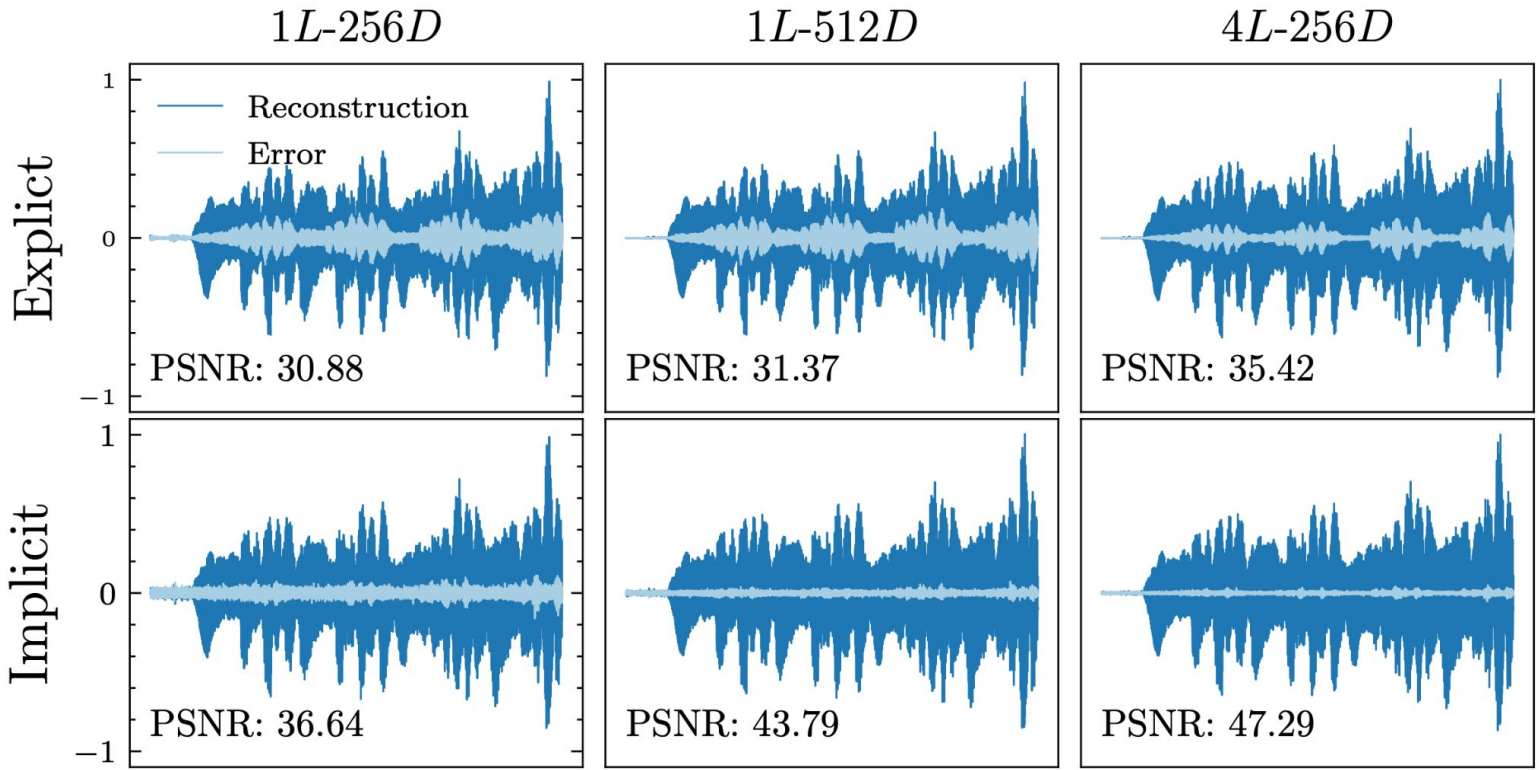
Image Generalization

- Train on only 25% of the pixels from each image and evaluate PSNR on an unobserved 25% portion of the image

	Natural			*Text*		
	$1L$-$256D$	$1L$-$512D$	$4L$-$256D$	$1L$-$256D$	$1L$-$512D$	$4L$-$256D$
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)	$24.88 \pm .44$	$\mathbf{25.19 \pm 3.64}$	24.52 ± 3.33	26.90 ± 2.14	$\mathbf{27.19 \pm 1.83}$	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	$\mathbf{25.42 \pm 3.76}$	24.53 ± 3.33	27.53 ± 2.18	$\mathbf{28.07 \pm 2.00}$	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	$\mathbf{24.92 \pm 3.58}$	24.05 ± 3.39	26.06 ± 2.18	$\mathbf{26.81 \pm 2.09}$	26.31 ± 2.20

Huang et al., “Implicit Layers for Implicit Representations” NeurIPS 2021.

Audio Representations



Huang et al., “Implicit Layers for Implicit Representations” NeurIPS 2021.

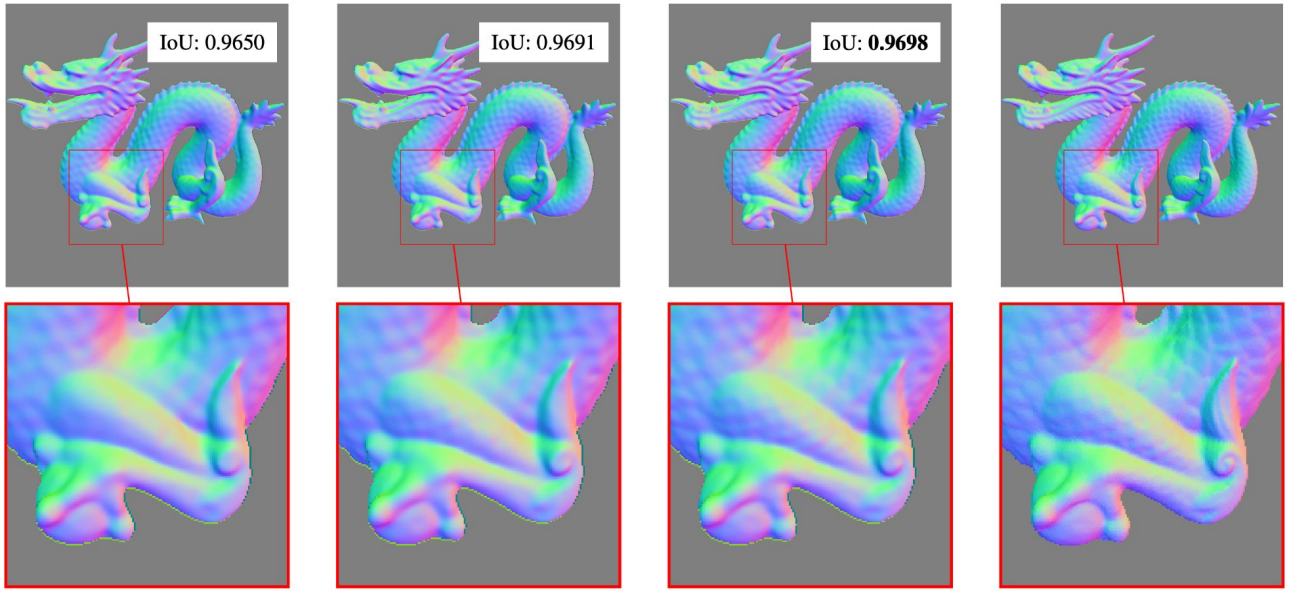
Video Representations

	$1L$-$1024D$	$1L$-$2048D$	$4L$-$1024D$
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	$\mathbf{28.03 \pm 0.95}$
Gabor-MFN	26.15 ± 1.02	28.18 ± 0.78	$\mathbf{29.64 \pm 0.81}$
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	$\mathbf{27.12 \pm 0.89}$

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

Huang et al., “Implicit Layers for Implicit Representations” NeurIPS 2021.

3D Geometry Representations

Explicit $1L$-$512D$	Explicit $4L$-$256D$	Implicit $1L$-$512D$	Ground Truth
			

Huang et al., “Implicit Layers for Implicit Representations” NeurIPS 2021.

Thank you!