Signal Processing for Implicit Neural Representations

Anonymous Author(s)
Affiliation
Address
email

Abstract

Implicit Neural Representations (INR) encoding continuous multi-media data via multi-layer perceptrons has shown undebatable promise in various computer vision tasks. Despite many successful applications, editing and processing an INR remains intractable as signals are represented by unintuitive parameters of a neural network. Existing works manipulate such continuous representations via processing on their discretized instance, which breaks down the compactness and continuous nature of INR. In this work, we present a pilot study on the question: *how to directly modify an INR without explicit decoding?* We answer this question by proposing an implicit neural signal processing network, dubbed **INSP-Net**, via differential operators on INR. Our key insight is that spatial gradients of neural networks can be computed analytically and are invariant to translation, while mathematically we show that any continuous convolution filter can be uniformly approximated by a linear combination of high-order differential operators. With these two knobs, we instantiate the INR signal operator as a composition of computational graphs corresponding to the high-order derivatives, where the weighting parameters can be either handcrafted or data-driven learned. Based on our proposed INSP-Net, we further build the first Convolutional Neural Network (CNN) that implicitly runs on INRs, named **INSP-ConvNet**. Our experiments validate the expressiveness of INSP-Net and INSP-ConvNet in fitting low-level image processing kernels (e.g. edge detection, blurring, deblurring, denoising, inpainting) as well as for high-level tasks on implicit fields such as image classification. We will release all codes.

1 Introduction

The idea that our visual world can be represented continuously has attracted increasing popularity in the field of implicit neural representations (INR). Also known as coordinate-based neural representations, INRs learn to encode a coordinate-to-value mapping for continuous multi-media data. Instead of storing the discrete signal values in a grid of pixels or voxels, INRs represent discrete data as samples of a continuous manifold. Using multi-layer perceptrons, INRs bring practical benefits to various computer vision applications, such as image and video compression [1, 2, 3], 3D shape representation [4, 5, 6, 7, 8, 9, 10, 11], inverse problems [12, 2, 13, 14], and generative models [15, 16, 17, 18, 19, 20, 21, 22].

Despite their recent success, INRs are not yet amenable to flexible editing and processing as the standard images could do. The encoded coordinate-to-value mapping is too complex to comprehend and the parameters stored in multi-layer perceptrons (MLPs) remains less explored. One direction of existing approaches enables editing on INRs by training them with conditional input. For example, [23, 24, 25, 20, 21, 26] utilize conditional codes to indicate different characteristics of the scene including shape and color. Another main direction benefits from existing image editing techniques and operates on discretized instances of continuous INRs such as pixels [27, 28] or voxels [29]. However, such solutions break down the continuous characteristic of INR and are not due to the prerequisite of decoding and discretizing before editing and processing.

Submitted to 36th Conference on Neural Information Processing Systems (NeurIPS 2022). Do not distribute.

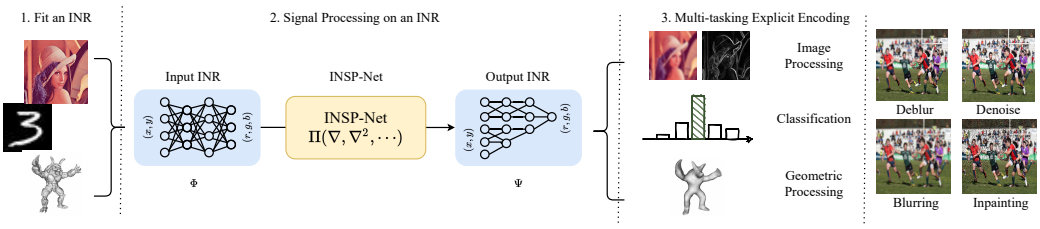


Figure 1: An illustration of implicit neural signal processing. Given an INR representing digital signals, our INSP-Net is capable of direct signal processing without needing to explicitly decode it. Our model first constructs derivative computation graphs of the original INR and then generates a linear combination of them into a new INR. It can be later decoded into discretized forms such as image pixels. The framework is capable of fitting low-level image processing kernels as well as performing high-level processing such as image classification.

In this paper, we conduct the first pilot study on the question: *how to generally modify an INR without explicit decoding?* The major challenge is that one cannot directly interpret what the parameters in an INR stand for, not to mention editing them correctly. Our key motivation is that spatial gradients can be served as a favorable tool to tackle this problem as they can be computed analytically, and possess desirable invariant properties. Theoretically, we prove that any continuous convolution filter can be uniformly approximated by a linear combination of high-order differential operators. Based on the above two rationales, we propose an Implicit Neural Signal Processing Network, dubbed **INSP-Net**, which processes INR utilizing high-order differential operators. The proposed INSP-Net is composed of an inception fusion block connecting computational graphs corresponding to derivatives of INRs. The weights in the branchy part are loaded from the INR being processing, while the weights in the fusion block are parameters of the represented operator, which can be either hand-crafted or learned by the data-driven algorithm. Even though we are not able to surgery on neural network parameters, we can implicitly process them by retrofitting their architecture and reorganizing the spatial gradients.

We further extend our framework to build the first Convolutional Neural Network (CNN) operating directly on INRs, dubbed INSP-ConvNet. Each layer of INSP-ConvNet is constructed by linearly combining the derivative computational graphs of the former layers. Nonlinear activation and normalization are naturally supported as they are element-wise functions. Data augmentation can be also implemented by augmenting the input coordinates of INRs. Under this pipeline (shown in Fig. 1), we demonstrate the expressiveness of our INSP-Net framework in fitting low-level image processing kernels including edge detection, blurring, deblurring, denoising, and image inpainting. We also successfully apply our INSP-ConvNet to high-level tasks on implicit fields such as classification.

Our main contributions can be summarized as follows:

- We propose a novel signal processing framework, dubbed **INSP-Net**, that operates on INRs analytically and continuously by closed-form high-order differential operators[1]. Repeatedly cascading the computational paradigm of INSP-Net, we also build a convolutional network, called INSP-ConvNet, which directly runs on implicit fields for high-level tasks.
- We illustrate the advantage of adopting differential operators by revealing their inherent group invariance. Furthermore, we rigorously prove that the convolution operator in the continuous regime can be uniformly approximated by a linear combination of the gradients.
- Extensive experiments demonstrate the effectiveness of our approach in both low-level processing (e.g. edge detection, blurring, deblurring, denoising, image inpainting) and high-level processing such as image classification.

2 Preliminaries: Implicit Neural Representation

Implicit Neural Representation (INR) parameterizes continuous multi-media signals or vector fields with neural networks. Formally, we consider an INR as a continuous function $\Phi : \mathbb{R}^m \to \mathbb{R}$ that maps low-dimension spatial/temporal coordinates to the value space[2]. For example, to represent 2D image

[1] By saying "closed-form", we mean the computation follows from an analytical mathematical expression.

[2] Without loss of generality, here we simplify Φ to be a scalar field, i.e., the range of Φ is one-dimensional.

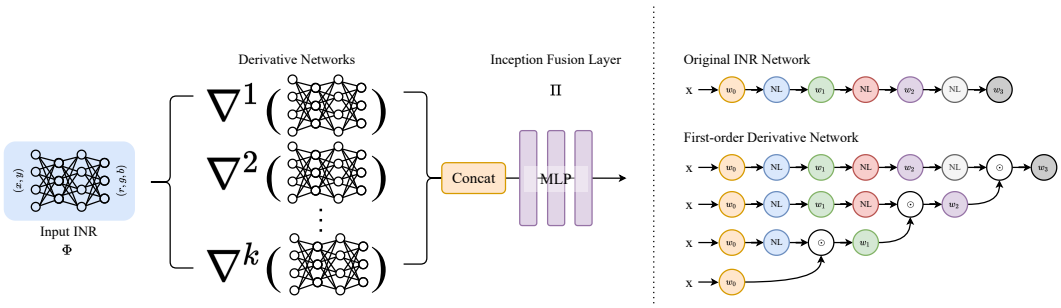


Figure 2: The left image provides an overview of our INSP-Net framework. Each layer combines the high-order derivative computational graphs of the original INR network. The right image illustrates the weight sharing scheme in calculating the derivative sub-networks.

signals, the domain of Φ is (x, y) spatial coordinates, and the range of Φ are the pixel intensities. The typical use of INR is to solve a feasibility problem where Φ is sought to satisfy a set of N constraints $\{\mathcal{C}(\Phi, a_m|\Omega_m)\}_{m=1}^N$, where $\mathcal{C}$ is a functional that relates function Φ to some observable quantities a_m evaluating over a measurable domain $\Omega_m \subseteq \mathbb{R}^m$. This problem can be cast into an optimization problem that minimizes deviations from each of the constraints:

$$\Phi^* = \arg\min_{\Phi} \sum_{m=1}^{N} \|\mathcal{C}(\Phi, a_m|\Omega_m)\|_2. \tag{1}$$

For instance, we can let $\mathcal{C} = \Phi(\boldsymbol{x}_m) - a_m$ with $\Omega_m = \{\boldsymbol{x}_m\}$, then our objective is reduced to a point-to-point supervision which memorizes a signal into Φ [30]. When functional $\mathcal{C}$ is a combination of differential operators taking values in a point set, i.e., $\mathcal{C}(a(\boldsymbol{x}), \Phi(\boldsymbol{x}), \nabla\Phi(\boldsymbol{x}), \cdots), \forall \boldsymbol{x} \in \Omega_m$, Eq. 1 is objective to solving a bunch of differential equations [27, 7, 31]. Note that in this paper, without particular specification, the gradients are all computed with respect to the input coordinate $\boldsymbol{x}$. $\mathcal{C}$ can also form an integral equation system over some intervals Ω_m [12]. In practice of computer vision, we reconstruct a signal by capturing sparse observations $\mathcal{D} = \{(\Omega_m, a_m)\}_{m=1}^N$ from unknown function Φ, and dynamically sampling a mini-batch from $\mathcal{D}$ to minimize Eq. 1 to obtain a feasible Φ.

A handy parameterization of function Φ is a fully-connected neural network, which enables solving Eq. 1 via gradient descent through a differentiable $\mathcal{C}$. Common INR networks consist of pure Multi-Layer Perceptrons (MLP) with periodic activation functions. Fourier Feature Mapping (FFM) [30] places a sinusoidal transformation before the MLP, while Sinusoidal Representation Network (SIREN) [27] replaces every piece-wise linear activation with a sinusoidal function. Below we give a unified formulation of INR networks:

$$\Phi(\boldsymbol{x}) = \boldsymbol{W}_n(\phi_{n-1} \circ \phi_{n-2} \circ \cdots \circ \phi_1)(\boldsymbol{x}), \quad \phi_i(\boldsymbol{x}) = \sigma_i(\boldsymbol{W}_i\boldsymbol{x} + \boldsymbol{b}_i), \tag{2}$$

where $\boldsymbol{W}_i \in \mathbb{R}^{d_{i-1}\times d_i}$, $\boldsymbol{b}_i \in \mathbb{R}^{d_i}$ are the weight matrix and bias of the i-th layer, respectively, n is the number of layers, and $\sigma_i(\cdot)$ is an element-wise nonlinear activation function. For FFM architecture, $\sigma_i = \sin(\cdot)$ when $i = 1$ denotes the positional encoding layer [12, 32] and otherwise $\sigma_i = \mathrm{ReLU}(\cdot)$. For SIREN, $\sigma_i = \sin(\cdot)$ for every layer $i = 1, \cdots, n-1$.

3 Implicit Representation Processing via Differential Operators

Digital Signal Processing (DSP) techniques have been widely applied in computer vision tasks, such as image restoration [33], signal enhancement [34] and geometric processing [35]. Even modern deep learning models are consisting of the most basic signal processing operators. Suppose we already acquire an Implicit Neural Representation (INR) $\Phi : \mathbb{R}^m \to \mathbb{R}$, now we are interested in whether we can run a signal processing program on the implicitly represented signals. One straightforward solution is to rasterize the implicit field with a 2D/3D lattice and run a typical kernel on the pixel/voxel grids. However, this decoding strategy produces a finite resolution and discretizes signals, which is memory inefficient and unfriendly to modeling fine details. In this section, we introduce a computation paradigm that can process an INR analytically with spatial/temporal derivatives. We show that our proposed method serves as a universal operator that can represent any continuous convolutional kernels.

3.1 Computational Paradigm

It has not escaped our notice that spatial/temporal gradients on INRs $\nabla^k \Phi$ can be computed analytically due to the differentiable characteristics of neural networks. Inspired by this, we propose an Implicit Neural Signal Processing (INSP) framework that composes a class of closed-form operators for INRs using functional combinations of high-order derivatives.

We denote our proposed signal processing operator by $\mathcal{A}$ built upon high-order derivatives. Given an acquired INR Φ, we denote the resultant INR processed by operator $\mathcal{A}$ as $\Psi = \mathcal{A}\Phi : \mathbb{R}^m \to \mathbb{R}$. To evaluate point $\boldsymbol{x} \in \mathbb{R}^m$ of processed INR, we propose the following computational paradigm:

$$\Psi(\boldsymbol{x}) := \mathcal{A}\Phi(\boldsymbol{x}) = \Pi\left(\Phi(\boldsymbol{x}), \nabla\Phi(\boldsymbol{x}), \nabla^2\Phi(\boldsymbol{x}), \cdots, \nabla^k\Phi(\boldsymbol{x}), \cdots\right), \tag{3}$$

where $\Pi : \mathbb{R}^M \to \mathbb{R}$ can be arbitrary continuous functions, which can be either handcrafted or learned from data. To learn an operator $\mathcal{A}$ from data, we represent Π by Multi-Layer Perceptrons (MLP) with parameters $\boldsymbol{\theta}$. Here we can slightly abuse the notation of ∇^k to be a flattened vector of high-order derivatives *without multiplicity* since differential operators defined over continuous functions form a commutative ring. The input dimension of Π depends on the highest order of used derivatives. Suppose we compute derivatives up to K-th order, then $M = \sum_{k=0}^{K} \binom{k+m-1}{k} = (K+1)\binom{K+m}{K+1}/m$, where $\binom{k+m-1}{k}$ is the number of distinctive k-th order differential operators [3]. As we will show in Sec. 3.2, Π can construct both shift-invariant and rotation-invariant operators, which introduces favorable inductive bias to images and 3D geometry processing. More importantly, we prove that Eq. 3 is also a universal approximator of arbitrary convolutional operators.

We note that $\Psi(\boldsymbol{x})$ as a whole can also be regarded as a neural network. Recall the architecture of $\Phi(\boldsymbol{x})$ in Eq. 2, its k-th order derivative is another computational graph parameterized by $\boldsymbol{W}_i$ and $\boldsymbol{b}_i$ that maps $\boldsymbol{x}$ to $\nabla^k\Phi(\boldsymbol{x})$. For example, the first-order gradient will have the following form:

$$\nabla\Phi(\boldsymbol{x}) = \hat{\phi}_{n-1} \circ (\phi_{n-2} \circ \cdots \circ \phi_1)(\boldsymbol{x}) \odot \cdots \odot \hat{\phi}_2 \circ \phi_1(\boldsymbol{x}) \odot \boldsymbol{W}_1 \tag{4}$$

where $\hat{\phi}_i(\boldsymbol{y}) = \boldsymbol{W}_i^\top \sigma'_{i-1}(\boldsymbol{W}_{i-1}\boldsymbol{y} + \boldsymbol{b}_{k-1})$, and $\sigma'_i(\cdot)$ is the first-order derivative of $\sigma_i(\cdot)$. Since $\hat{\phi}_i$ shares the weights with ϕ_i, $\nabla\Phi$ is represented by a closed-form computational network re-using the weights from Φ, which we refer to as the first-order *derivative network*. The higher-order derivatives should induce the derivative network of similar forms. Therefore, the processed INR Ψ will have an Inception-like architecture, namely, a multi-branch structure connecting the original INR network and weight-sharing derivative subnetworks followed by a fusion layer Π. We call the entire model ($\Psi = \mathcal{A}\Phi$ or Eq. 3) an *Implicit Neural Signal Processing Network* or an *INSP-Net*. Note that the only parameters of INSP-Net $\boldsymbol{\theta}$ are located at the last fusion layer, and can be trained in an end-to-end manner.

We illustrate an INSP-Net in Fig. 2 where the color indicates the weight-sharing scheme. A similar weight-sharing scheme is also adopted in AutoInt [36]. In practice, we employ auto-differentiation in PyTorch [37] to automatically create such derivatives networks and reassemble them parallelly to constitute the architecture of an INSP-Net. When inputting an INR, we load the weights of the INR to our model following the weight-sharing scheme, and then we obtain an INSP-Net, which implicitly and continuously represents the processed INR $\Psi(\boldsymbol{x})$. To effectively express high-order derivatives, we choose SIREN as the base model [27].

3.2 Theoretical Analysis

In this section, we provide a theoretical justification for the design of our INSP-Net. We will focus on discussing the latent invariance property and the expressive power of INSP-Net.

Translation and Rotation Invariance. Group invariance has testified as a favorable inductive bias for image [38], video [39], and geometry processing [40]. It has also been well-known that group invariance is an intrinsic property of Partial Differential Equations (PDEs) [41, 42]. Since our INSP-Net is built using differential operators, we are motivated to reveal its hidden invariance property to demonstrate its advantage in processing visual signals.

In this section, we only consider two transformation groups: translation group $\mathbb{T}(m)$ and the special orthogonal group $\mathbb{SO}(m)$ (a.k.a. rotation group). Elements $T_{\boldsymbol{v}} \in \mathbb{T}(m)$ in translation group shifts the

[3] This is equal to the number of monic monomials over $\mathbb{R}^m$ with degree k.

function Φ by some offset $\boldsymbol{v} \in \mathbb{R}^m$. The shifted function can be denoted as $\Phi \circ T_{\boldsymbol{v}}(\boldsymbol{x}) = \Phi(\boldsymbol{x} + \boldsymbol{v})$. Similarly, elements in rotation group perform a coordinate transformation on function Φ by a rotation matrix $\boldsymbol{R} \in \mathbb{SO}(m)$. The transformed function can be written as $\Phi \circ \boldsymbol{R}(\boldsymbol{x}) = \Phi(\boldsymbol{R}\boldsymbol{x})$. Group invariance means deforming the input space of a function first and then processing it via an operator is equivalent to directly applying the transformation to the processed function. For a more rigorous argument, $\mathcal{A}$ is said to be translation-invariant if $\forall T_{\boldsymbol{v}} \in \mathbb{T}(m)$, $\Psi(\boldsymbol{x} + \boldsymbol{v}) = \mathcal{A}[\Phi \circ T_{\boldsymbol{v}}](\boldsymbol{x})$. Likewise, $\mathcal{A}$ is rotation-invariant if $\forall \boldsymbol{R} \in \mathbb{SO}(m)$ we have $\Psi(\boldsymbol{R}\boldsymbol{x}) = \mathcal{A}[\Phi \circ \boldsymbol{R}](\boldsymbol{x})$. Below we provide Theorem 1 to characterize the invariance property for our model.

Theorem 1. *Given function $\Pi : \mathbb{R}^M \to \mathbb{R}$, the composed operator $\mathcal{A}$ (Eq. 3) can satisfy:*

1. *shift invariance for every Π.*
2. *rotation invariance if Π has the form: $\Pi(\boldsymbol{y}) = f(\|\boldsymbol{y}\|_2)$ for some $f : \mathbb{R} \to \mathbb{R}$.*

We prove Theorem 1 in Appendix A. Our Theorem 1 implies that operator $\mathcal{A}$ is inherently shift-invariant. This is due to the shift-invariant intrinsics of differential operators as we show in the proof. Rotation invariance is not guaranteed in general. However, if one carefully designs Π, it can also be achieved via our framework. Moreover, we also suggest a feasible solution to constructing a rotation-invariant operator $\mathcal{A}$ in Theorem 1. In our construction, Π first isotropically pools over the squares of all directional derivatives, and then maps the summarized information through another function f, which can be either handcrafted or a trainable neural network. We refer interested readers [41] for more group invariance in differential forms.

Universal Approximation. Convolution, formally known as the linear shift-invariant operator, has served as one of the most prevalent signal processing tools in the vision domain. Given two (real-valued) signals f and g, we denote their convolution as $g \star f = f \star g$. In this section, we examine the expressiveness of our INSP-Net (Eq. 3) by showing it can represent any convolutional filter. To draw this conclusion, we first present an informal version of our main results as follows:

Theorem 2. *(Informal statement) For every real-valued function $g : \mathbb{R} \to \mathbb{R}$, there exists a polynomial $p(x) = a_0 + a_1 x + \cdots$ where $a_0, a_1, \cdots \in \mathbb{R}$, such that $p(\nabla) f$ can uniformly approximate $g \star f$ by arbitrary precision for all real-valued signals f.*

The formal statement and proof can be found in Appendix B. Theorem 2 involves the notion of polynomials in differential operators. We define $p(\nabla) = a_0 + a_1 \nabla + a_2 \nabla^2 + \cdots$, where the map that takes polynomial $p(x)$ to $p(\nabla)$ is a ring homomorphism from polynomial ring $\mathbb{R}[x]$ to the ring of endomorphism of a function space. Theorem 2 can be regarded as an application of the Stone-Weierstrass approximation theorem on the Fourier domain. However, we note that functions obtained by the Fourier transform are generally complex functions. The prominence of our results is that we can constrain the range of the polynomial coefficients into the real domain, which makes it implementable via a common deep learning infrastructure. The implication of Theorem 2 is that the mapping between convolution and derivative is as simple as a linear transformation. Recent works [43, 44, 45] show similar results that derivatives can be computed via a linear combination of discrete convolution. Theorem 2 can be served as their converse argument but in the continuous regime. In our proof, k-th order derivatives correspond to k-th order monomial function in spectral space. This indicates that differential operators form a basis of the linear shift-invariant operator space. In other words, any convolution can be written as a linear combination of differential operators, i.e., $p(\nabla)$. Since a linear function is not difficult to be approximated by a neural network $\Pi_{\boldsymbol{\theta}}$, we can derive the next straightforward result Corollary 3.

Corollary 3. *For every real-valued function g, there exists a neural network $\Pi_{\boldsymbol{\theta}}$ such that $\Psi = \mathcal{A}\Phi$ (Eq. 3) can uniformly approximate $g \star \Phi$ by arbitrary precision for every real-valued signals Φ.*

As we discussed in Theorem 1, $\mathcal{A}$ are constantly shift-invariant. This means when approximating a convolutional kernel, the trajectory of $\mathcal{A}$ is restricted into the shift-invariant space. Moreover, we emphasize that INSP-Net is far more expressive than convolutional kernels since $\Pi_{\boldsymbol{\theta}}$ can also fit any nonlinear continuous functions due to the universal approximation theorem [46, 47].

3.3 Building CNNs for Implicit Neural Representations

Convolutional Neural Networks (CNN) are capable of extracting informative semantics by only piling up basic signal processing operators. This motivates us to build CNNs based on INSP-Net that can

directly run on INRs for high-level downstream tasks. In fact, to simulate *exact* convolution, our Theorem 2 suggests simplify $\Pi_{\boldsymbol{\theta}}$ to a linear mapping. Then our former computational paradigm Eq. 3 is changed to:

$$\Psi(\boldsymbol{x}) := p(\nabla)\Phi(\boldsymbol{x}) = \theta_0\Phi(\boldsymbol{x}) + \boldsymbol{\theta}_1^\top \nabla\Phi(\boldsymbol{x}) + \boldsymbol{\theta}_2^\top \nabla^2\Phi(\boldsymbol{x}) + \cdots + \boldsymbol{\theta}_k^\top \nabla^k\Phi(\boldsymbol{x}) + \cdots, \tag{5}$$

where $\boldsymbol{\theta}_k \in \mathbb{R}^{\binom{k+m-1}{k}}$ are parameters of the operator $p(\nabla)$. We name this special case of Eq. 3 as INSP-Conv. One plausible implementation of INSP-Conv is to employ a one-layer MLP to represent $\Pi_{\boldsymbol{\theta}}$. When $\mathcal{A} = p(\nabla)$, INSP-Conv preserves both linearity and shift-invariance when evolving during the training. We propose to repeatedly apply INSP-Conv with non-linearity to INRs that mimics a CNN-like architecture. We name this class of CNNs composed by multi-layer INSP-Conv (Eq. 5) as INSP-ConvNet. Previous works [48, 49] extracting semantic features from INR either lack local information by point-wisely mapping INR's intermediate representation to a semantic space or explicitly rasterize INR into regular grids. To the best of our knowledge, it is the first time that one can run a CNN directly on an implicit representation thanks to closed-formness of INSP-Net. The overall architecture of INSP-ConvNet can be formulated as:

$$\mathrm{ConvNet}[\Phi](\boldsymbol{x}) = \mathcal{A}^{(L)} \cdot \sigma \circ \mathcal{A}^{(L-1)} \cdot \sigma \circ \cdots \circ \mathcal{A}^{(2)} \cdot \sigma \circ \mathcal{A}^{(1)} \cdot \Phi(\boldsymbol{x}), \tag{6}$$

where σ is an element-wise non-linear activation, L is the number of INSP-Net layers, and Φ is an input INR. We use the symbol $\cdot$ to denote operator functioning, and $\circ$ to denote function composition. Due to page limit, we defer detailed introduction to INSP-ConvNet to Appendix C.

4 Related Works

4.1 Implicit Neural Representation

Implicit Neural Representation (INR) represents signals by continuous functions parameterized by multi-layer perceptrons (MLPs) [27, 30], which is different from traditional discrete representations (e.g., pixel, mesh). Compared with other representations, the continuous implicit representations are capable of representing signals theoretically at infinite resolution and have become prevailing to be applied upon image fitting [27], image compression [1, 50] and video compressing [3]. In addition, INR has been applied to more efficient and effective shape representation [4, 5, 6, 7, 8, 9, 10, 11], texture mapping [51, 52], inverse problems [12, 2, 13, 14] and generative models [15, 16, 17, 18, 19, 20, 21, 22]. There are also efforts speeding up the fitting of INRs [53] and improving the representation efficiency [54]. Nowadays, editing and manipulating multi-media objects gains increasing interest and demand [55]. Thus, signal processing on implicit neural representation is essentially an important task worth investigating.

4.2 Editable Implicit Fields

Editing implicit fields has recently attracted much research interest. Several methods have been proposed to allow editing the reconstructed 3D scenes by rearranging the objects or manipulating the shape and appearance. One line of work alters the structure and color of objects by conditioning latent codes for different characteristics of the scene [25, 20, 21, 26]. Another direction involves discretizing the continuous fields. By converting the implicit fields into pixels [27, 28] or voxels [29], traditional image and voxel editing techniques can be applied effortlessly. Recently, NFGP [56] proposes to use neural fields for geometry processing. These approaches, however, are not capable of directly performing signal processing on continuous INRs. Our INSP-Net makes smart use of closed-form differential operators and thus does not require explicit decoding.

4.3 PDE based Image Processing

Partial differential equations (PDEs) have been successfully applied to many tasks in image processing and computer vision, such as image enhancement [57, 58, 59], segmentation [60, 42], image registration [61], saliency detection [62] and optical flow computation [63]. Early traditional PDEs are written directly based on mathematical and physical understanding of the PDEs (e.g., anisotropic diffusion [57], shock filter [58] and curve evolution based equations [64, 65, 66]). Variational design methods [59, 67, 66] start from an energy function describing the desired properties of output images and compute the Euler-Lagrange equation to derive the evolution equations. Learning-based attempts [42, 62] build PDEs from image pairs based on the assumption (without proof) that PDEs

could be written as linear combinations of fundamental differential invariants. Although it might be feasible to let INRs solve this bunch of signal processing PDEs, one needs to per-case re-fit an INR with an additional temporal axis, which is severely sampling inefficient. The multi-layer structure appearing in INSP-Net can be viewed as an unfolding algorithm [68, 69] that discretizes the evolution of time-variant PDEs [70].

5 Experiments

In this section, we evaluate the proposed **INSP** framework on several challenging tasks, using different combinations of Π. First, we build low-level image processing filters using either hand-crafted or learnable Π. Then, we construct convolutional neural networks with our INSP-ConvNet framework and validate its performance on image classification. More results and implementation details are provided in the Appendix.

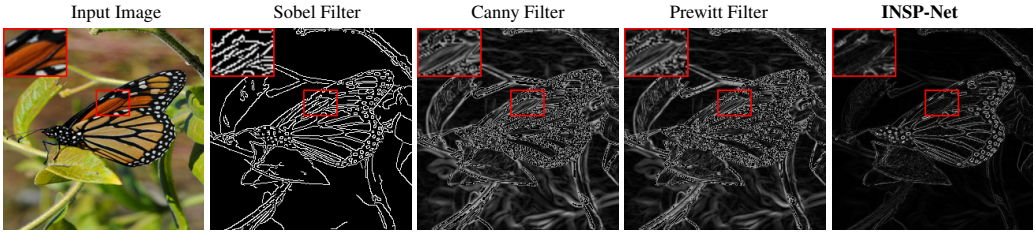


Figure 3: Edge detection. We fit the natural images with SIREN and use our INSP-Net to process implicitly into a new INR that can be decoded into edge maps.

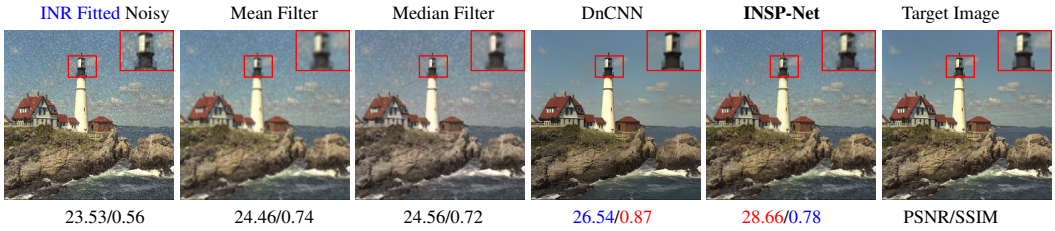


Figure 4: Image denoising. We fit the noisy images with SIREN and train our INSP-Net to process implicitly into a new INR that can be decoded into natural clear images.

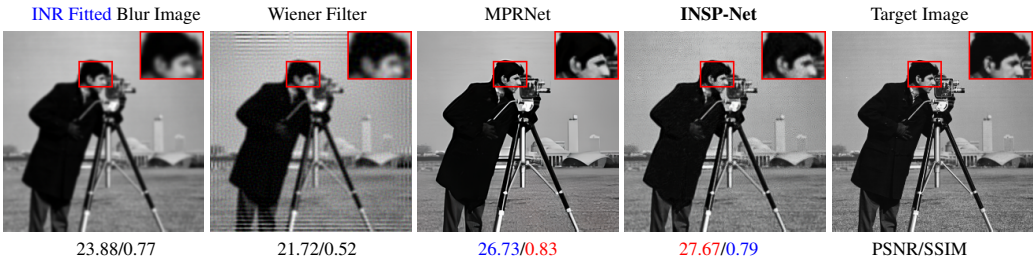


Figure 5: Image deblurring. We fit the blurred images with SIREN and train our INSP-Net to process implicitly into a new INR that can be decoded into clear natural images.

For low-level image processing, we operate on natural images from Set5 dataset [71], Set14 dataset [72], and DIV-2k dataset [73]. Originally designed for super-resolution, the images are diverse in style and content. Note that the unprocessed images presented are the images decoded from unprocessed INRs.

5.1 Handcrafted Π for Edge Detection

We demonstrate the proposed **INSP** framework is capable of expressing low-level image processing filters such as for edge detection. Since the edges correspond to gradients in the images, using

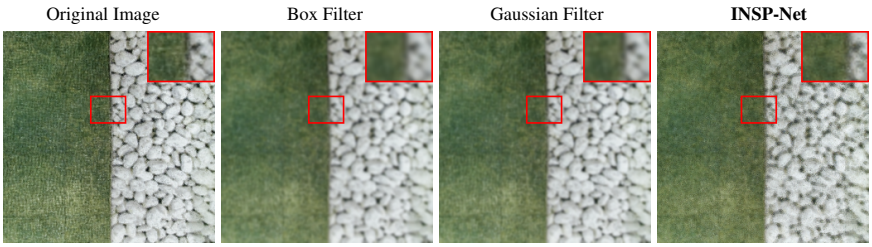


Figure 6: Image blurring. We fit the natural images with SIREN and train our INSP-Net to process implicitly into a new INR that can be decoded into blurred images.

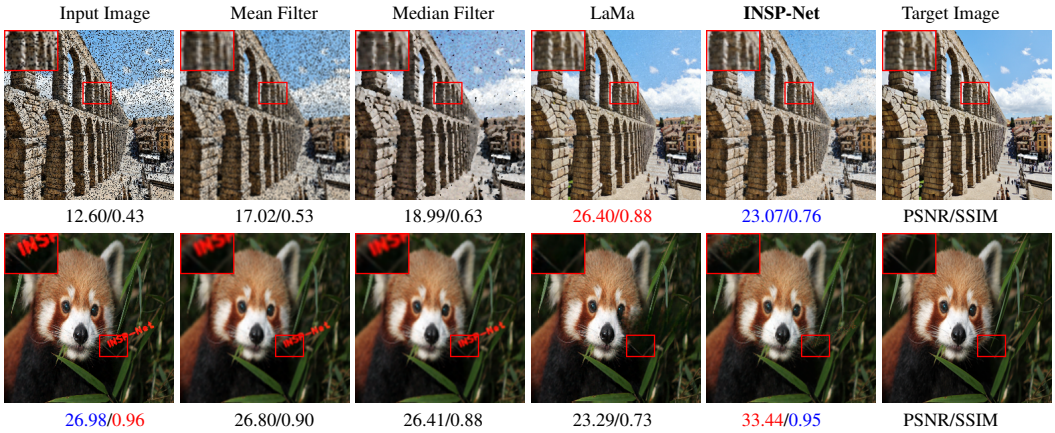


Figure 7: Image inpainting. We fit the input images with SIREN and train our INSP-Net to process implicitly into a new INR that can be decoded into natural images. Note that LaMa requires explicit masks to select the regions for inpainting and the masks are roughly provided.

gradients of INRs to obtain edges is straightforward. θ_1 is set to 1 while other coefficients are set to 0. In this way, we implicitly obtain a new INR which can be decoded to obtain the edges. We provide visual comparisons against Sobel filter [74], Canny detector [75] and Prewitt operator [76] in Fig. 3.

5.2 Learnable Π for Image Blurring, Denoising, Deblurring, and Inpainting

Since our method operates directly on INRs, we firstly fit the images with INRs and then feed the INRs into our framework. The final output is another INR which can be decoded into desired images. The training set of our method consists of 90 examples of INRs, where each INR is built on SIREN [27] architectures.

Image Denoising For classical image denoising filters, we compare against the median filter and mean filter. We use DnCNN [77] as a baseline learning-based method. DnCNN is a pioneer work of using residual CNN for denoising. The input noisy images are synthesized using additive gaussian noise. Visual results are provided in Fig. 4.

Image Blurring Image blurring is a low-pass filtering operation. We provide a visual comparison against classical filters including 3×3 box filter and 3×3 gaussian filter. The target images used for training our INSP-Net are the results of the Gaussian filter. Visual results are provided in Fig. 6.

Image Deblurring We compare the proposed method with both traditional algorithms (e.g., wiener filter [78]) and learning-based algorithms(e.g., MPRNet [79]). We synthesize blurry images using Gaussian filters. As shown in Fig. 5, Wiener Filter produce severe artifacts and MPRNet successfully reconstructs clear textures. INSP-Net is capable of generating competitive results against MPRNet and outperforms the Wiener Filter.

Image Inpainting We conduct two kinds of experiments in image inpainting, to inpaint 30% random masked pixels or to remove the texts ("INSP-Net"). Comparison methods include mean filter, median filter, and LaMa [80]. LaMa is a learning-based method using Fourier convolution

for inpainting. As shown in Fig. 7, mean filter and median filter partially restore the masked pixels, but severely hurt the visual quality of the rest parts. Also, they can not handle the text region. LaMa successfully removes the text and inpaint the masked pixels. Our proposed method largely outperforms the filter-based algorithms and performs as well as the LaMa.

5.3 Learnable Π for Geometry Processing

We demonstrate that the proposed **INSP** framework is not only capable of processing images, but also capable of processing geometry. We first fit a SIREN using the unprocessed mesh following [25]. Then we use our INSP-Net to implicitly process it into a new INR. As shown in Fig. 8, our method is able to smooth the surface of the mesh.

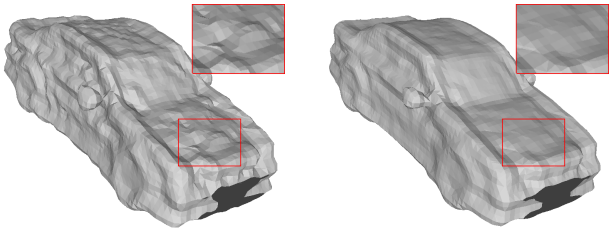


Figure 8: Left: Unprocessed geometry decoded from an unprocessed INR. Right: Smoothed geometry decoded from the output INR of our INSP-Net.

5.4 Convolutional Neural Networks for Image Classification

We demonstrate that the proposed **INSP** framework is not only capable to express low-level image processing filters, but also supports high-level tasks such as image classification. To achieve this goal, we construct a 2-layer INSP-ConvNet. The INSP-ConvNet consists of 2 INSP-Net layers. Each of them decomposes the INR via the differential operator and combines them with learnable Π. Since each INSP-Net layer combines the derivative computational graphs of the former layers and approximates a convolution filter, we build another 2-layer depthwise ConvNets running on pixels as the baseline, for a fair comparison.

We evaluate the proposed INSP-ConvNet on MNIST (28×28 resolution) and CIFAR-10 (32×32 resolution) datasets, respectively. For each dataset, we will firstly fit each image into an implicit representation using SIREN [27]. Both experiments take 1000 epochs to optimize, using AdamW optimizer [81] and a learning rate of 10^{-4}. Results are shown in Tab. 1.

Accuracy	Depthwise CNN	INSP-ConvNet
MNIST	87.6%	88.1%
CIFAR-10	59.5%	62.5%

Table 1: Quantitative Results of Image Classification.

6 Conclusion

Contribution. We present INSP-Net framework, an implicit neural signal processing network that is capable of directly modifying an INR without explicit decoding. By incorporating differential operators on INR, we can instantiate the INR signal operator as a composition of computational graphs approximating any continuous convolution filter. Furthermore, we make the first effort to build a convolutional neural network that implicitly runs on INRs. While all other methods run on discrete grids, our experiment demonstrates our INSP-Net can achieve competitive results with entirely implicit operations.

Limitation. (Theory) Our theory only guarantees the expressiveness of convolution by allowing infinite sequence approximation. Construction of more expressive operators and more effective parameterization for convolution remain widely open questions. (Practice) INSP-Net requires the computation of high-order derivatives which is neither memory efficient nor numerical stable. This hinders the scalability of our INSP-ConvNet that even requires recursive computation of derivatives.

References

[1] Emilien Dupont, Adam Goliński, Milad Alizadeh, Yee Whye Teh, and Arnaud Doucet. Coin: Compression with implicit neural representations. *arXiv preprint arXiv:2103.03123*, 2021.

[2] Hao Chen, Bo He, Hanyu Wang, Yixuan Ren, Ser Nam Lim, and Abhinav Shrivastava. Nerv: Neural representations for videos. *Advances in Neural Information Processing Systems*, 34, 2021.

[3] Yunfan Zhang, Ties van Rozendaal, Johann Brehmer, Markus Nagel, and Taco Cohen. Implicit neural video compression. *arXiv preprint arXiv:2112.11312*, 2021.

[4] Kyle Genova, Forrester Cole, Daniel Vlasic, Aaron Sarna, William T Freeman, and Thomas Funkhouser. Learning shape templates with structured implicit functions. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 7154–7164, 2019.

[5] Matan Atzmon, Niv Haim, Lior Yariv, Ofer Israelov, Haggai Maron, and Yaron Lipman. Controlling neural level sets. *Advances in Neural Information Processing Systems*, 32, 2019.

[6] Zhiqin Chen and Hao Zhang. Learning implicit fields for generative shape modeling. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5939–5948, 2019.

[7] Amos Gropp, Lior Yariv, Niv Haim, Matan Atzmon, and Yaron Lipman. Implicit geometric regularization for learning shapes. *arXiv preprint arXiv:2002.10099*, 2020.

[8] Lars Mescheder, Michael Oechsle, Michael Niemeyer, Sebastian Nowozin, and Andreas 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.

[9] Michael Niemeyer, Lars Mescheder, Michael Oechsle, and Andreas Geiger. Occupancy flow: 4d reconstruction by learning particle dynamics. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 5379–5389, 2019.

[10] Jeong Joon Park, Peter Florence, Julian Straub, Richard Newcombe, and Steven Lovegrove. Deepsdf: Learning continuous signed distance functions for shape representation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 165–174, 2019.

[11] Songyou Peng, Michael Niemeyer, Lars Mescheder, Marc Pollefeys, and Andreas Geiger. Convolutional occupancy networks. In *European Conference on Computer Vision*, pages 523–540. Springer, 2020.

[12] Ben Mildenhall, Pratul P Srinivasan, Matthew Tancik, Jonathan T Barron, Ravi Ramamoorthi, and Ren Ng. Nerf: Representing scenes as neural radiance fields for view synthesis. In *European conference on computer vision*, pages 405–421. Springer, 2020.

[13] Vincent Sitzmann, Semon Rezchikov, William T Freeman, Joshua B Tenenbaum, and Fredo Durand. Light field networks: Neural scene representations with single-evaluation rendering. *arXiv preprint arXiv:2106.02634*, 2021.

[14] Ben Mildenhall, Peter Hedman, Ricardo Martin-Brualla, Pratul Srinivasan, and Jonathan T Barron. Nerf in the dark: High dynamic range view synthesis from noisy raw images. *arXiv preprint arXiv:2111.13679*, 2021.

[15] Eric R Chan, Marco Monteiro, Petr Kellnhofer, Jiajun Wu, and Gordon Wetzstein. pi-gan: Periodic implicit generative adversarial networks for 3d-aware image synthesis. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 5799–5809, 2021.

[16] Terrance DeVries, Miguel Angel Bautista, Nitish Srivastava, Graham W Taylor, and Joshua M Susskind. Unconstrained scene generation with locally conditioned radiance fields. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 14304–14313, 2021.

[17] Jiatao Gu, Lingjie Liu, Peng Wang, and Christian Theobalt. Stylenerf: A style-based 3d-aware generator for high-resolution image synthesis. *arXiv preprint arXiv:2110.08985*, 2021.

[18] Zekun Hao, Arun Mallya, Serge Belongie, and Ming-Yu Liu. Gancraft: Unsupervised 3d neural rendering of minecraft worlds. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 14072–14082, 2021.

[19] Quan Meng, Anpei Chen, Haimin Luo, Minye Wu, Hao Su, Lan Xu, Xuming He, and Jingyi Yu. Gnerf: Gan-based neural radiance field without posed camera. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 6351–6361, 2021.

[20] Michael Niemeyer and Andreas Geiger. Giraffe: Representing scenes as compositional generative neural feature fields. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11453–11464, 2021.

[21] Katja Schwarz, Yiyi Liao, Michael Niemeyer, and Andreas Geiger. Graf: Generative radiance fields for 3d-aware image synthesis. *Advances in Neural Information Processing Systems*, 33:20154–20166, 2020.

[22] Peng Zhou, Lingxi Xie, Bingbing Ni, and Qi Tian. Cips-3d: A 3d-aware generator of gans based on conditionally-independent pixel synthesis. *arXiv preprint arXiv:2110.09788*, 2021.

[23] Steven Liu, Xiuming Zhang, Zhoutong Zhang, Richard Zhang, Jun-Yan Zhu, and Bryan Russell. Editing conditional radiance fields. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 5773–5783, 2021.

[24] Can Wang, Menglei Chai, Mingming He, Dongdong Chen, and Jing Liao. Clip-nerf: Text-and-image driven manipulation of neural radiance fields. *arXiv preprint arXiv:2112.05139*, 2021.

[25] Jeong Joon Park, Peter Florence, Julian Straub, Richard Newcombe, and Steven Lovegrove. Deepsdf: Learning continuous signed distance functions for shape representation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 165–174, 2019.

[26] Zhiqin Chen and Hao Zhang. Learning implicit fields for generative shape modeling. *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5932–5941, 2019.

[27] Vincent Sitzmann, Julien Martel, Alexander Bergman, David Lindell, and Gordon Wetzstein. Implicit neural representations with periodic activation functions. *Advances in Neural Information Processing Systems*, 33:7462–7473, 2020.

[28] James W. Hennessey, Wilmot Li, Bryan Russell, Eli Shechtman, and Niloy J. Mitra. Transferring image-based edits for multi-channel compositing. *ACM Transactions on Graphics*, 36(6), 2017.

[29] Jerry Liu, Fisher Yu, and Thomas Funkhouser. Interactive 3d modeling with a generative adversarial network. In *2017 International Conference on 3D Vision (3DV)*, pages 126–134. IEEE, 2017.

[30] Matthew Tancik, Pratul P Srinivasan, Ben Mildenhall, Sara Fridovich-Keil, Nithin Raghavan, Utkarsh Singhal, Ravi Ramamoorthi, Jonathan T Barron, and Ren Ng. Fourier features let networks learn high frequency functions in low dimensional domains. *arXiv preprint arXiv:2006.10739*, 2020.

[31] Jiequn Han, Arnulf Jentzen, and E Weinan. Solving high-dimensional partial differential equations using deep learning. *Proceedings of the National Academy of Sciences*, 115(34):8505–8510, 2018.

[32] Ellen D Zhong, Tristan Bepler, Bonnie Berger, and Joseph H Davis. Cryodrgn: reconstruction of heterogeneous cryo-em structures using neural networks. *Nature Methods*, 18(2):176–185, 2021.

[33] Dejia Xu, Yihao Chu, and Qingyan Sun. Moiré pattern removal via attentive fractal network. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, pages 472–473, 2020.

[34] Yaqian Xu, Wenqing Zheng, Jingchen Qi, and Qi Li. Blind image blur assessment based on markov-constrained fcm and blur entropy. In *2019 IEEE international conference on image processing (icip)*, pages 4519–4523. IEEE, 2019.

[35] Peng-Shuai Wang, Xiao-Ming Fu, Yang Liu, Xin Tong, Shi-Lin Liu, and Baining Guo. Rolling guidance normal filter for geometric processing. *ACM Transactions on Graphics (TOG)*, 34(6):1–9, 2015.

[36] David B Lindell, Julien NP Martel, and Gordon Wetzstein. Autoint: Automatic integration for fast neural volume rendering. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 14556–14565, 2021.

[37] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019.

[38] Jan J Koenderink and Andrea J van Doorn. Image processing done right. In *European Conference on Computer Vision*, pages 158–172. Springer, 2002.

[39] Hae-Kwang Kim and Jong-Deuk Kim. Region-based shape descriptor invariant to rotation, scale and translation. *Signal Processing: Image Communication*, 16(1-2):87–93, 2000.

[40] Niloy J Mitra, Mark Pauly, Michael Wand, and Duygu Ceylan. Symmetry in 3d geometry: Extraction and applications. In *Computer Graphics Forum*, volume 32, pages 1–23. Wiley Online Library, 2013.

[41] Peter J Olver. *Applications of Lie groups to differential equations*, volume 107. Springer Science & Business Media, 2000.

[42] Risheng Liu, Zhouchen Lin, Wei Zhang, and Zhixun Su. Learning pdes for image restoration via optimal control. In *European Conference on Computer Vision*, pages 115–128. Springer, 2010.

[43] Bin Dong, Qingtang Jiang, and Zuowei Shen. Image restoration: Wavelet frame shrinkage, nonlinear evolution pdes, and beyond. *Multiscale Modeling & Simulation*, 15(1):606–660, 2017.

[44] Zichao Long, Yiping Lu, Xianzhong Ma, and Bin Dong. Pde-net: Learning pdes from data. In *International Conference on Machine Learning*, pages 3208–3216. PMLR, 2018.

[45] Zichao Long, Yiping Lu, and Bin Dong. Pde-net 2.0: Learning pdes from data with a numeric-symbolic hybrid deep network. *Journal of Computational Physics*, 399:108925, 2019.

[46] George Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals and Systems*, 1989.

[47] Kurt Hornik. Approximation capabilities of multilayer feedforward networks. *Neural Networks*, 1991.

[48] Shuaifeng Zhi, Tristan Laidlow, Stefan Leutenegger, and Andrew J Davison. In-place scene labelling and understanding with implicit scene representation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 15838–15847, 2021.

[49] Suhani Vora, Noha Radwan, Klaus Greff, Henning Meyer, Kyle Genova, Mehdi SM Sajjadi, Etienne Pot, Andrea Tagliasacchi, and Daniel Duckworth. Nesf: Neural semantic fields for generalizable semantic segmentation of 3d scenes. *arXiv preprint arXiv:2111.13260*, 2021.

[50] Brandon Yushan Feng and Amitabh Varshney. Signet: Efficient neural representation for light fields. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 14224–14233, 2021.

[51] Michael Oechsle, Lars Mescheder, Michael Niemeyer, Thilo Strauss, and Andreas Geiger. Texture fields: Learning texture representations in function space. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 4531–4540, 2019.

[52] Shunsuke Saito, Zeng Huang, Ryota Natsume, Shigeo Morishima, Angjoo Kanazawa, and Hao Li. Pifu: Pixel-aligned implicit function for high-resolution clothed human digitization. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 2304–2314, 2019.

[53] Matthew Tancik, Ben Mildenhall, Terrance Wang, Divi Schmidt, Pratul P Srinivasan, Jonathan T Barron, and Ren Ng. Learned initializations for optimizing coordinate-based neural representations. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2846–2855, 2021.

[54] Jaeho Lee, Jihoon Tack, Namhoon Lee, and Jinwoo Shin. Meta-learning sparse implicit neural representations. *Advances in Neural Information Processing Systems*, 34:11769–11780, 2021.

[55] Emilien Dupont, Hyunjik Kim, SM Eslami, Danilo Rezende, and Dan Rosenbaum. From data to functa: Your data point is a function and you should treat it like one. *arXiv preprint arXiv:2201.12204*, 2022.

[56] Guandao Yang, Serge Belongie, Bharath Hariharan, and Vladlen Koltun. Geometry processing with neural fields. *Advances in Neural Information Processing Systems*, 34, 2021.

[57] Pietro Perona and Jitendra Malik. Scale-space and edge detection using anisotropic diffusion. *IEEE Transactions on pattern analysis and machine intelligence*, 12(7):629–639, 1990.

[58] Stanley Osher and Leonid I Rudin. Feature-oriented image enhancement using shock filters. *SIAM Journal on numerical analysis*, 27(4):919–940, 1990.

[59] Xue-Cheng Tai, Stanley Osher, and Randi Holm. Image inpainting using a tv-stokes equation. In *Image Processing based on partial differential equations*, pages 3–22. Springer, 2007.

[60] Zhouchen Lin, Wei Zhang, and Xiaoou Tang. Designing partial differential equations for image processing by combining differental invariants, 2009.

[61] Lars Hömke, Claudia Frohn-Schauf, Stefan Henn, and Kristian Witsch. Total variation based image registration. In *Image processing based on partial differential equations*, pages 343–361. Springer, 2007.

[62] Risheng Liu, Junjie Cao, Zhouchen Lin, and Shiguang Shan. Adaptive partial differential equation learning for visual saliency detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3866–3873, 2014.

[63] Adam Rabcewicz. Clg method for optical flow estimation based on gradient constancy assumption. In *Image Processing Based on Partial Differential Equations*, pages 57–66. Springer, 2007.

[64] Guillermo Sapiro. *Geometric partial differential equations and image analysis*. Cambridge university press, 2006.

[65] Frédéric Cao. *Geometric curve evolution and image processing*. Springer Science & Business Media, 2003.

[66] Bart M Haar Romeny. *Geometry-driven diffusion in computer vision*, volume 1. Springer Science & Business Media, 2013.

[67] Leonid I Rudin, Stanley Osher, and Emad Fatemi. Nonlinear total variation based noise removal algorithms. *Physica D: nonlinear phenomena*, 60(1-4):259–268, 1992.

[68] Karol Gregor and Yann LeCun. Learning fast approximations of sparse coding. In *Proceedings of the 27th international conference on international conference on machine learning*, pages 399–406, 2010.

[69] Jialin Liu and Xiaohan Chen. Alista: Analytic weights are as good as learned weights in lista. In *International Conference on Learning Representations (ICLR)*, 2019.

[70] Ricky TQ Chen, Yulia Rubanova, Jesse Bettencourt, and David K Duvenaud. Neural ordinary differential equations. *Advances in neural information processing systems*, 31, 2018.

[71] Marco Bevilacqua, Aline Roumy, Christine Guillemot, and Marie line Alberi Morel. Low-complexity single-image super-resolution based on nonnegative neighbor embedding. In *Proceedings of the British Machine Vision Conference*, pages 135.1–135.10. BMVA Press, 2012.

[72] Roman Zeyde, Michael Elad, and Matan Protter. On single image scale-up using sparse-representations. In *International conference on curves and surfaces*, pages 711–730. Springer, 2010.

[73] Eirikur Agustsson and Radu Timofte. Ntire 2017 challenge on single image super-resolution: Dataset and study. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, July 2017.

[74] I Sobel. An isotropic $3\times$ 3 image gradient operator, presentation at stanford ai project (1968), 2014.

[75] John Canny. A computational approach to edge detection. *IEEE Transactions on pattern analysis and machine intelligence*, (6):679–698, 1986.

[76] Judith MS Prewitt et al. Object enhancement and extraction. *Picture processing and Psychopictorics*, 10(1):15–19, 1970.

[77] Kai Zhang, Wangmeng Zuo, Yunjin Chen, Deyu Meng, and Lei Zhang. Beyond a gaussian denoiser: Residual learning of deep cnn for image denoising. *IEEE transactions on image processing*, 26(7):3142–3155, 2017.

[78] Saeed V Vaseghi. *Advanced digital signal processing and noise reduction*. John Wiley & Sons, 2008.

[79] Syed Waqas Zamir, Aditya Arora, Salman Khan, Munawar Hayat, Fahad Shahbaz Khan, Ming-Hsuan Yang, and Ling Shao. Multi-stage progressive image restoration. In *CVPR*, 2021.

[80] Roman Suvorov, Elizaveta Logacheva, Anton Mashikhin, Anastasia Remizova, Arsenii Ashukha, Aleksei Silvestrov, Naejin Kong, Harshith Goka, Kiwoong Park, and Victor Lempitsky. Resolution-robust large mask inpainting with fourier convolutions. *arXiv preprint arXiv:2109.07161*, 2021.

[81] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.

[82] François Chollet. Xception: Deep learning with depthwise separable convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1251–1258, 2017.

[83] Zhengzhong Tu, Hossein Talebi, Han Zhang, Feng Yang, Peyman Milanfar, Alan Bovik, and Yinxiao Li. Maxim: Multi-axis mlp for image processing. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5769–5780, 2022.

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] Please see the conclusion section.
   (c) Did you discuss any potential negative societal impacts of your work? [No]
   (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? [Yes]
   (b) Did you include complete proofs of all theoretical results? [Yes] The detailed proof is provided 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)? [No]
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] More details are provided in the Appendix.
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [No]
   (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] More details are provided in the Appendix. All experiments are run on a A6000 GPU.
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] We cite them carefully.
   (b) Did you mention the license of the assets? [No] The data are in public domain.
   (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? [N/A]
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]
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]

A Proof of Theorem 1

To begin with, we give the formal definitions of translation and rotation group, along with the notion of shift invariance and rotation invariance.

Definition 1. *(Translation Group & Shift Invariance) Translation group $\mathbb{T}(m)$ is a transformation group isomorphic to m-dimension Euclidean space, where each group element $T_{\boldsymbol{v}}$ transforms a vector $\boldsymbol{x} \in \mathbb{R}^m$ by $T_{\boldsymbol{v}}(\boldsymbol{x}) = \boldsymbol{x} + \boldsymbol{v}$. An operator $\mathcal{A}$ is said to be shift-invariant if $\mathcal{A}(\Phi \circ T_{\boldsymbol{v}})(\boldsymbol{x}) = \mathcal{A}\Phi(T_{\boldsymbol{v}}(\boldsymbol{x})) = \mathcal{A}\Phi(\boldsymbol{x} + \boldsymbol{v})$.*

Definition 2. *(Rotation Group & Rotation Invariance) Rotation group $\mathbb{SO}(m)$ is a transformation group also known as the special orthogonal group, where each group element $\boldsymbol{R} \in \mathbb{R}^{m\times m}$ satisfying $\boldsymbol{R}^\top \boldsymbol{R} = \boldsymbol{I}$ transforms a vector $\boldsymbol{x} \in \mathbb{R}^m$ by $\boldsymbol{R}(\boldsymbol{x})$. An operator $\mathcal{A}$ is said to be rotation-invariant if $\mathcal{A}(\Phi \circ \boldsymbol{R})(\boldsymbol{x}) = \mathcal{A}\Phi(\boldsymbol{R}\boldsymbol{x})$.*

Proof. (Shift Invariance) To show the shift invariance of our model Eq. 3, it is equivalent to show any differential operators are shift-invariant. For the first-order derivatives (gradients), we consider arbitrary shift operator $T_{\boldsymbol{v}} \in \mathbb{T}$, by chain rule we will have:

$$\nabla[\Phi \circ T_{\boldsymbol{v}}](\boldsymbol{x}) = \left[\frac{d(\boldsymbol{x}+\boldsymbol{v})}{d\boldsymbol{x}}\right]^\top \nabla\Phi(\boldsymbol{x}+\boldsymbol{v}) = \nabla\Phi(\boldsymbol{x}+\boldsymbol{v}), \tag{7}$$

where the Jacobian matrix of $T_{\boldsymbol{v}}(\boldsymbol{x})$ is an identity matrix. Eq. 7 implies that gradient operator is shift-invariant. By induction, any high-order differential operators must also be shift-invariant:

$$\nabla^k[\Phi \circ T_{\boldsymbol{v}}](\boldsymbol{x}) = \nabla^k\Phi(\boldsymbol{x}+\boldsymbol{v}), \tag{8}$$

Therefore, we can conclude $\Pi(\Phi, \nabla\Phi, \nabla^2\Phi, \cdots)$ is shift-invariant for any Π combining derivatives in any form.

(Rotation Invariance) Similarly, we prove the rotation invariance via induction. First we consider the following case: given arbitrary function $\Phi : \mathbb{R}^m \to \mathbb{R}$, and for every rotation matrix $\boldsymbol{R} \in \mathbb{SO}(m)$, we compute the gradient:

$$\nabla[\Phi \circ \boldsymbol{R}](\boldsymbol{x}) = \boldsymbol{R}^\top \nabla\Phi(\boldsymbol{R}\boldsymbol{x}). \tag{9}$$

Then due to the orthogonality of $\boldsymbol{R}$, the norm of all the gradients can be written as:

$$\|\nabla[\Phi \circ \boldsymbol{R}](\boldsymbol{x})\|_F^2 = \mathrm{Tr}\left[\nabla\Phi(\boldsymbol{R}\boldsymbol{x})^\top \boldsymbol{R}\boldsymbol{R}^\top \nabla\Phi(\boldsymbol{R}\boldsymbol{x})\right] = \|\nabla\Phi(\boldsymbol{R}\boldsymbol{x})\|_F^2. \tag{10}$$

The induction can be proceed by assuming $\|\nabla^k[\Phi \circ \boldsymbol{R}](\boldsymbol{x})\|_2^2 = \|\nabla^k\Phi(\boldsymbol{R}\boldsymbol{x})\|_2^2$, where with slight abuse of notation, we flatten the k-th order Jacobian matrix to a $\binom{k+m-1}{k}$-dimension vector without multiplicity. Then for the $(k+1)$-th order Jacobian matrix (represented as an m by $\binom{k+m-1}{k}$ matrix), we have

$$\nabla^{k+1}[\Phi \circ \boldsymbol{R}](\boldsymbol{x}) = \nabla^k \cdot \nabla[\Phi \circ \boldsymbol{R}](\boldsymbol{x}) = \nabla^k[\boldsymbol{R}^\top \circ \nabla\Phi \circ \boldsymbol{R}](\boldsymbol{x}). \tag{11}$$

Then by the induction hypothesis, we can obtain:

$$\begin{aligned}
\left\|\nabla^k[\boldsymbol{R}^\top \circ \nabla\Phi \circ \boldsymbol{R}](\boldsymbol{x})\right\|_F^2 &= \left\|\nabla^k[\boldsymbol{R}^\top \circ \nabla\Phi](\boldsymbol{R}\boldsymbol{x})\right\|_F^2 = \left\|\nabla^{k+1}\Phi(\boldsymbol{R}\boldsymbol{x})^\top \boldsymbol{R}\right\|_F^2 && (12)\\
&= \mathrm{Tr}\left[\boldsymbol{R}^\top \nabla^{k+1}\Phi(\boldsymbol{R}\boldsymbol{x})\nabla^{k+1}\Phi(\boldsymbol{R}\boldsymbol{x})^\top \boldsymbol{R}\right] && (13)\\
&= \mathrm{Tr}\left[\nabla^{k+1}\Phi(\boldsymbol{R}\boldsymbol{x})\nabla^{k+1}\Phi(\boldsymbol{R}\boldsymbol{x})^\top\right] = \|\nabla^{k+1}\Phi(\boldsymbol{R}\boldsymbol{x})\|_F^2. && (14)
\end{aligned}$$

where Eq. 12 is due to the linearity of differential operators. As we show in Eq. 10, $k = 0$ satisfies the induction hypothesis, then we can draw the conclusion: for arbitrary differentiable function Φ and for every integer $k > 0$, operator $\|\nabla^k\Phi(\boldsymbol{x})\|_2^2$ is rotation-invariant. Hence, $\Pi = f\left(\left\|\begin{bmatrix}\Phi(\boldsymbol{x}) & \nabla\Phi(\boldsymbol{x}) & \nabla^2\Phi(\boldsymbol{x}) & \cdots\end{bmatrix}\right\|_F\right) = f\left(\sqrt{\sum_{k=0}\|\nabla^k\Phi(\boldsymbol{x})\|_2^2}\right)$ is also rotation-invariant as for arbitrary f mapping the L_2 norm of derivatives. □

B Proof of Theorem 2

To give a formal statement of Theorem 2, we first list the following definitions:

Definition 3. *(Differential Operator) For a sake of simplicity and consistency, we denote $\mathcal{D} = \frac{d}{dx}[\cdot]$ as the first-order differential operator, $\mathcal{D}^n = \frac{d^n}{dx^n}[\cdot]$ as the n-th order differential operator.*

Definition 4. *(Fourier Transform) Given real-valued function $f : \mathbb{R} \to \mathbb{C}$ that satisfies Dirichlet condition[4], then Fourier transform $\mathcal{F}$ is defined as:*

$$\mathcal{F}[f](\omega) = \int_{-\infty}^{\infty} f(x) \exp(-2\pi i \omega x) dx. \tag{15}$$

Inverse Fourier transform $\mathcal{F}^{-1}$ exists and has the form of:

$$f(x) = \int_{-\infty}^{\infty} \mathcal{F}[f](\omega) \exp(2\pi i \omega x) d\omega. \tag{16}$$

Definition 5. *(Convolution) Given two real-valued functions $f : \mathbb{R} \to \mathbb{R}$ and $g : \mathbb{R} \to \mathbb{R}$, convolution between f and g is defined as:*

$$(f \star g)(x) = \int_{-\infty}^{+\infty} f(x - \xi) g(\xi) d\xi. \tag{17}$$

Then we denote function $f \star g = \mathcal{T}_g[f]$, where $\mathcal{T}_g$ represents a functional operator with respect to g.

Definition 6. *(Band-limited Signal Space) Define the signal space $\mathcal{F}$ as a Hilbert space of real-valued functions such that for $\forall f \in \mathcal{F}$, (1) $f \in C_\infty(\mathbb{R})$ is continuous and smooth over real, (2) $\int_{-\infty}^{\infty} |f(x)| dx < \infty$ is L_1 integrable, (3) $\mathcal{F}[f](\omega) = 0$ if $|\omega| > a$ has a limited width of spectrum, and $\int_{-\infty}^{\infty} |\mathcal{F}[f](\omega)| d\omega \leq C$ has limited spectral power, for some $a > 0$, $C > 0$.*

Definition 7. *(Convolution Space) Define a convolutional operator space $\mathcal{T}$ of all $\mathcal{T}_g$ such that $\forall \mathcal{T}_g \in \mathcal{T}$, (1) $g : \mathbb{R} \to \mathbb{R}$ is real-valued function, (2) $\mathcal{F}[g] \in C_1(\mathbb{R})$ has a continuous spectrum.*

Before we prove Theorem 2, we enumerate the following results as our key mathematical tools:

Lemma 4. *(Convolution Theorem) For every $\mathcal{T}_g \in \mathcal{T}$, it always holds that $\mathcal{F} \circ \mathcal{T}_g[f](\omega) = \mathcal{F}[f](\omega) \cdot \mathcal{F}[g](\omega)$.*

Lemma 5. *(Weierstrass Approximation Theorem) Suppose f is a continuous real-valued function defined on a compact space $\mathcal{X}$. For every $\epsilon > 0$, there exists a polynomial $p(x)$ such that $\sup_{x \in \mathcal{X}} |f(x) - p(x)| < \epsilon$.*

Lemma 6. *Suppose f is a continuous real-valued function satisfying Dirichlet condition. Then $\mathcal{F}[f](\omega) = \mathcal{F}[f](-\omega)^*$, i.e., the spectrum of real-valued function is conjugate symmetric.*

Below is the formal statement of our Theorem 2 and its detailed proof.

Theorem 7. *For every $\mathcal{T}_g \in \mathcal{T}$ and arbitrary small $\epsilon > 0$, there exists a polynomial $p(x) = a_0 + a_1 x + \cdots$ where $a_0, a_1, \cdots \in \mathbb{R}$, such that $\sup_{x \in \mathbb{R}} |\mathcal{T}_g[f](x) - p(\mathcal{D})[f](x)| < \epsilon$ for all $f \in \mathcal{F}$.*

Proof. For every $f \in \mathcal{F}$ and $\mathcal{T}_g \in \mathcal{T}$, by Lemma 4, one can rewrite:

$$\mathcal{F} \circ \mathcal{T}_g[f](\omega) = \mathcal{F}[f](\omega) \cdot \mathcal{F}[g](\omega) := \hat{f}(\omega)\hat{g}(\omega), \tag{18}$$

where we use $\hat{f} : \mathbb{R} \to \mathbb{C}$ and $\hat{g} : \mathbb{R} \to \mathbb{C}$ to denote the Fourier transform of f and g, respectively. By Lemma 6, we can construct an invertible mapping ϕ by letting:

$$\phi[\hat{f}](w) = \Re\{\hat{f}(w)\} - \Im\{\hat{f}(w)\}, \tag{19}$$

$$\phi^{-1}[\tilde{f}](w) = \frac{\tilde{f}(w) + \tilde{f}(-w)}{2} - i\frac{\tilde{f}(w) - \tilde{f}(-w)}{2}, \tag{20}$$

which is also known as the Hartley transform. Then $\tilde{f} := \phi[\hat{f}]$ and $\tilde{g} := \phi[\hat{g}]$ turn out to be real-valued functions. Since $\hat{f}$ is only supported in $[-a, a]$, we only consider $\tilde{g}$ within the interval $[-a, a]$. By

[4] This guarantees Fourier transform exists.

Lemma 5, there exists a polynomial $\tilde{p}(x) = \tilde{a}_0 + \tilde{a}_1 x + \cdots + \tilde{a}_d x^d$ where $d = \deg(\tilde{p})$ is the degree of $\tilde{p}$ and $\tilde{a}_0, \tilde{a}_1, \cdots, \tilde{a}_d \in \mathbb{R}$ such that $\sup_{\omega \in [-a,a]} |\tilde{g}(\omega) - \tilde{p}(\omega)| < \epsilon/2C$ for every $\epsilon > 0$.

Applying ϕ^{-1} to $\tilde{p}$, we will obtain a new polynomial $\hat{p} := \phi^{-1}[\tilde{p}]$ over $\mathbb{C}$ such that:

$$\Re\{\hat{p}(\omega)\} = \frac{\tilde{p}(w) + \tilde{p}(-w)}{2}, \qquad \Im\{\hat{p}(\omega)\} = \frac{\tilde{p}(w) - \tilde{p}(-w)}{2}. \tag{21}$$

Then we show the difference between $\hat{g}$ and $\hat{p}$:

$$\begin{aligned}
|\hat{g}(\omega) - \hat{p}(\omega)| = &\left| \left(\frac{\tilde{f}(w) + \tilde{f}(-w)}{2} - \frac{\tilde{p}(w) + \tilde{p}(-w)}{2} \right) \right. && (22)\\
&\left. - i \left(\frac{\tilde{f}(w) - \tilde{f}(-w)}{2} - \frac{\tilde{p}(w) - \tilde{p}(-w)}{2} \right) \right| && (23)\\
\leq &\frac{1}{2} \left(\left| \tilde{f}(w) - \tilde{p}(w) \right| + \left| \tilde{f}(-w) - \tilde{p}(-w) \right| \right) && (24)\\
&+ \frac{1}{2} \left(\left| \tilde{p}(w) - \tilde{f}(w) \right| + \left| \tilde{p}(-w) - \tilde{f}(-w) \right| \right) && (25)\\
\leq &\frac{\epsilon}{C}. && (26)
\end{aligned}$$

Hereby, we have shown that we can construct a complex polynomial $\hat{p}(x) = \hat{a}_0 + \hat{a}_1 x + \cdots + \hat{a}_d$ from the real polynomial $\tilde{p}(x) = \tilde{a}_0 + \tilde{a}_1 x + \cdots + \tilde{a}_d$ by letting $\hat{a}_{2k} = \tilde{a}_{2k}, k = 0, 1, \cdots, \lfloor \frac{d}{2} \rfloor$ and $\hat{a}_{2k+1} = i\tilde{a}_{2k+1}, k = 0, 1, \cdots, \lfloor \frac{d-1}{2} \rfloor$, such that polynomial $\hat{p}$ satisfies that $\sup_{\omega \in [-a,a]} |\hat{g}(\omega) - \hat{p}(\omega)| < \epsilon/C$.

In the meanwhile, $f \in \mathcal{F}$ is L_1 integrable over $\mathbb{R}$ implies $\lim_{x \to \infty} f(x) = 0$. Then by the fundamental theorem of calculus, we have the following equation:

$$\begin{aligned}
0 = f(x) \exp(-2\pi i \omega x) \Big|_{-\infty}^{+\infty} &= \int_{-\infty}^{+\infty} \frac{d}{dx} \left[f(x) \exp(-2\pi i \omega x) \right] dx && (27)\\
&= \int_{-\infty}^{+\infty} \left[\frac{df(x)}{dx} \exp(-2\pi i \omega x) - 2\pi i \omega f(x) \exp(-2\pi i \omega x) \right] dx && (28)\\
&= \mathcal{F}\left[\frac{df(x)}{dx} \right](\omega) - (2\pi i \omega) \cdot \mathcal{F}[f(x)](\omega), && (29)
\end{aligned}$$

which implies $\mathcal{F} \circ \mathcal{D}[f](\omega) = 2\pi i \omega \cdot \mathcal{F}[f](\omega)$. Define a sequence $q_n(\omega) = (2\pi i \omega)^n$. By induction, the n-th order derivative of f satisfies

$$\mathcal{F} \circ \mathcal{D}^n[f](\omega) = q_n(\omega) \cdot \mathcal{F}[f](\omega). \tag{30}$$

We note that $\{q_n, n = 0, 2, \cdots,\}$ form a basis of the polynomial ring, where q_{2k} only has real part, q_{2k+1} only has imaginary part, for $k = 0, 1, \cdots$. Thus, we can find a series of $a_0, a_1, \cdots, a_d \in \mathbb{R}$ such that $\hat{p}(x) = a_0 + a_1 q_1(\omega) + \cdots + a_d q_d(\omega)$ by letting $a_n = \hat{a}_n/(2\pi i)^n \in \mathbb{R}, n = 0, 1, \cdots, d$. So far, we obtain $a_0, a_1, \cdots, a_d \in \mathbb{R}$ such that $\sup_{\omega \in [-a,a]} |\hat{g}(\omega) - \sum_{n=0}^{d} a_n q_n(\omega)| < \epsilon/C$.

By linearity of Fourier transform and Eq. 30, element-wisely applying $\sum_{n=0}^{d} a_n q_n$ to $\hat{f}$ will lead to a transform on the real domain:

$$\mathcal{F}^{-1}\left[\sum_{n=0}^{d} a_n q_n \cdot \mathcal{F}[f] \right] = \sum_{n=0}^{d} a_n \mathcal{D}^n f := p(\mathcal{D})[f], \tag{31}$$

Algorithm 1 Forward pass of INSP-ConvNet

1: **Input**: An INR network $\Phi(\boldsymbol{x}) : \mathbb{R}^m \to \mathbb{R}$, convolutional operator weights $\boldsymbol{\theta}^{(l)} \in \mathbb{R}^M$ and an input coordinate $\boldsymbol{x}$.
2: **Output**: Value at $\boldsymbol{x}$ of INR $\mathrm{ConvNet}[\Phi]$ processed by INSP-ConvNet.
3: $y^{(0)} \leftarrow \Phi(\boldsymbol{x})$
4: **for** $l = 1, \cdots, L$ **do**
5: $\quad \hat{y}^{(l)} \leftarrow \left[y^{(l-1)} \quad \frac{\partial y^{(l-1)}}{\partial \boldsymbol{x}}^\top \quad \frac{\partial^2 y^{(l-1)}}{\partial \boldsymbol{x}^2}^\top \quad \cdots \quad \frac{\partial^K y^{(l-1)}}{\partial \boldsymbol{x}^K}^\top \right] \boldsymbol{\theta}^{(l)}$ ▷ Convolutional layer
6: $\quad y^{(l)} \leftarrow \mathrm{ReLU}(\mathrm{InstanceNorm1D}(\hat{y}^{(l)}))$ ▷ Non-linearity and normalization
7: **end for**
8: **return** $y^{(L)}$.

where $p(x) = a_0 + a_1 x + \cdots + a_d x^d$ over ring of differentiation operators. Now we bound the difference between $\mathcal{T}_g[f]$ and $p(\mathcal{D})[f]$ for every $f \in \mathcal{F}$ and $x \in \mathbb{R}$:

$$|\mathcal{T}_g[f](x) - p(\mathcal{D})[f](x)| = \left| \int_{-a}^{a} \exp(2\pi i \omega x) \hat{f}(\omega) \left(\hat{g}(\omega) - \sum_{n=0}^{d} a_n q_n(\omega) \right) d\omega \right| \tag{32}$$

$$\leq \int_{-a}^{a} \left| \hat{f}(\omega) \left(\hat{g}(\omega) - \sum_{n=0}^{d} a_n q_n(\omega) \right) \right| d\omega \tag{33}$$

$$\leq \left(\sup_{\omega \in [-a,a]} \left| \hat{g}(\omega) - \sum_{n=0}^{d} a_n q_n(\omega) \right| \right) \left(\int_{-a}^{a} \left| \hat{f}(\omega) \right| d\omega \right) \tag{34}$$

$$\leq \epsilon, \tag{35}$$

where Eq. 33 holds due to Jensen's inequality, Eq. 34 follows from Hölder's inequality, and Eq. 35 is obtained by substituting the upper bound of difference $|\hat{g}(\omega) - \sum_{n=0}^{d} a_n q_n(\omega)|$ and the L_1 norm of $\hat{f}(\omega)$. □

C Implementation Details of INSP-ConvNet

We have formulated exact convolution form and INSP-Conv in Sec. 3.3. We provide a pseudocode to illustrate the forward pass of INSP-ConvNet in Algorithm 1. Below we elaborate each main component:

Convolutional Layer. Each $\mathcal{A}^{(l)}$ represents an implicit convolution layer. We follow the closed-form solution in Eq. 5 to parameterize $\mathcal{A}^{(l)}$ with $\boldsymbol{\theta}^{(l)}$. We point out that $\mathrm{ConvNet}[\Phi]$ also corresponds to a computational graph, which can continuously map coordinates to the output features. To construct this computational graph, we recursively call for gradient networks of the previous layer until the first layer. For example, $\mathcal{A}^{(l)}$ will request the gradient network of $\mathcal{A}^{(l-1)} \cdot \sigma \circ \cdots \circ \mathcal{A}^{(1)} \cdot \Phi$, and then $\mathcal{A}^{(l-1)}$ will request the gradient network of the rest part. This procedure will proceed until the first layer, which directly returns the derivative network of Φ. Kernels in CNNs typically perform multi-channel convolution. However, this is not memory friendly to gradient computing in our framework. To this end, we run channel-wise convolution first and then employ a linear layer to mix channels [82].

Nonlinear Activation and Normalization. Nonlinear activation and normalization are naturally element-wise functions. They are point-wisely applied to the output of an INSP-Net and participate the computational graph construction process. This corresponds to the line 6 of Algorithm 1.

Training Recipe. Given a dataset $\mathcal{D} = \{(\Phi_i, y_i)\}$ with a set of pre-trained INRs Φ_i and their corresponding labels y_i, our goal is to learn a $\mathrm{ConvNet}[\cdot]$ that can process each example. In contrast to standard ConvNets that are designed for grid-based images, the computational graph of INSP-ConvNet contains parameters of both the input INR Φ_i and learnable kernels $\mathcal{A}^{(l)}$. During the training stage, we randomly sample a mini-batch (Φ_i, y_i) from $\mathcal{D}$ to optimize INSP-ConvNet. The corresponding loss will be evaluated according to the network output, and then back-propagate

the calculated gradients to the learnable parameters in $\mathcal{A}^{(l)}$, using the stochastic gradient descent optimization. Along the whole process, the parameters of Φ_i is fixed and only the parameters in $\mathcal{A}^{(l)}$ is optimized. Standard data augmentations are included by default, including rotation, zoom in/out, etc. In practice, we implement these augmentations by using affine transformation on the coordinates of INRs.

D Connection with PDE based Signal Processing

Partial Differential Equation (PDE) has been successfully applied to image processing domain as we discussed in Sec. 4.3. In this section, we focus on their connection with our INSP-Net. We summarize the methods of this line of works [57, 42, 62] in the following formulation:

$$\frac{\partial \Psi(\boldsymbol{x}, t)}{\partial t} = M_t \left[\Psi(\boldsymbol{x}, t), \nabla_{\boldsymbol{x}} \Psi(\boldsymbol{x}, t), \nabla^2_{\boldsymbol{x}} \Psi(\boldsymbol{x}, t), \cdots \right], \tag{36}$$

where $M_t(\cdot)$ is a time-variant function that remaps the direct output and high-order derivatives of function Ψ. For heat diffusion, M_t boils down to be an stationary isotropic combination of second-order derivatives. In [57], M_t is chosen to be a gradient magnitude aware diffusion operator running on divergence operators. [42, 62] degenerate M_t to a time-dependent linear mapping of pre-defined invariants of the maximal order two. We note that Eq. 36 can be naturally solved with INRs, as INRs are amenable to solving complicated differential equation shown by [27]. One straightforward solution is to parameterize M_t by another time-dependent coordinate network [30] and enforce the boundary condition $\Psi(\boldsymbol{x}, 0) = \Phi(\boldsymbol{x})$ and minimize the difference between the two hands of the Eq. 36. However, foreseeable problem falls in sampling inefficiency over the time axis. Suppose we discretize the time axis into small intervals $0 = t_0 < t_1 < \cdots < t_N$, then Eq. 36 has a closed-form solution given M_t:

$$\Psi(\boldsymbol{x}, t_{n+1}) = \int_{t_n}^{t_{n+1}} M_t \left[\Psi(\boldsymbol{x}, t), \nabla_{\boldsymbol{x}} \Psi(\boldsymbol{x}, t), \nabla^2_{\boldsymbol{x}} \Psi(\boldsymbol{x}, t), \cdots \right] dt + \Psi(\boldsymbol{x}, t_n) \tag{37}$$

$$\approx M_{t_n} \left[\Psi(\boldsymbol{x}, t_n), \nabla_{\boldsymbol{x}} \Psi(\boldsymbol{x}, t_n), \nabla^2_{\boldsymbol{x}} \Psi(\boldsymbol{x}, t_n), \cdots \right] (t_{n+1} - t_n) + \Psi(\boldsymbol{x}, t_n). \tag{38}$$

One can see Eq. 38 can be regarded as a special case of our model Eq. 3, where we absorb M_{t_n}, time interval $t_{n+1} - t_n$, and the residual term $\Psi(\boldsymbol{x}, t_n)$ into one Π. Considering our multi-layer model INSP-ConvNet (see Sec. 3.3), we can analogize t_n to the layer number, and then solving Eq. 36 at time t_N is approximately equal to forward passing an N-layer INSP-ConvNet.

E More Experiment Details

We implement our INSP framework using PyTorch. The gradients are obtained directly using the autograd package from PyTorch. All learnable parameters are trained with AdamW optimizer and a learning rate of 1e-4. For low-level image processing kernels, images are obtained from Set5 dataset [71], Set14 dataset [72], and DIV-2k dataset [73]. These datasets are original collected for super-resolution task, so the images are diverse in style and content. In our experiments, we construct SIREN [27] on each image. For efficiency, we resized the images to 256×256. We use 90 images to construct the INRs used for training, and use the other images for evaluation.

For image classification, we construct a 2-layer INSP-ConvNet framework. Each INSP layer constructs the derivative computational graphs of the former layers and combines them with learnable Π. The INSP-layer is capable of approximating a convolution filter. For a fair comparison, we build another 2-layer depthwise convolutional network running on image pixels as the baseline. Both our INSP-ConvNet and the ConvNet running on pixels are trained with the same hyper-parameters. Both experiments take 1000 training epochs, with a learning rate of $1e-4$ using AdamW optimizer.

F More Experimental Results

In this section, we provide more experimental results. Fig. 9 provides comparisons on edge detection task. Fig. 10 shows image denoising results. Fig. 11 demonstrates image deblurring results. Fig. 12 shows image blurring results. Fig. 13 shows image inpainting results.

	PSNR	SSIM	LPIPS
Input (decoded from INR)	23.23	0.64	0.33
MPRNet [79]	21.70	0.73	0.32
MAXIM [83]	24.32	0.80	0.26
INSP-Net	**27.95**	**0.82**	**0.23**

Table 2: Quantitative result of image denoising on 100 testing images from DIV-2k dataset [73], where the synthetic noise is gray gaussian noise clipped to be positive. MPRNet and MAXIM overfit to the SIDD dataset they were trained on, and don't generalize well on the synthetic Gaussian noise we are using here.

	PSNR	SSIM	LPIPS
Input (decoded from INR)	20.51	0.47	0.40
MPRNet [79]	23.95	0.72	0.36
MAXIM [83]	24.64	0.74	0.33
INSP-Net	23.86	0.65	0.38

Table 3: Quantitative result of image denoising on 100 testing images from DIV-2k dataset [73], where the synthetic noise is rgb gaussian noise. The noise is similar to the ones seen during the training of MPRNet and MAXIM, so they obtain better performance with the help of a much wider training set.

We also provide quantitative comparisons on the test set in Tab. 2 and Tab. 3. The test set consists of 100 INRs fitted from 100 images in DIV-2k dataset [73]. In Tab. 2, the synthetic noise used is a gray-scale Gaussian noise. We additionally clip the value of the noise so that only the positive part remains. Under such situations, MPRNet [79] and MAXIM [83] do not generalize well because the synthetic noise differs from the SIDD dataset they were trained on. However, in Tab. 3, their performance is better when the synthetic noise becomes three-channel Gaussian noise. The synthetic noise is similar to those seen during the training process of MPRNet [79] and MAXIM [83], so they benefit from their much wider training set.

We additionally validate the ability of our INSP framework by processing audio signals. We add synthetic Gaussian noise onto the audio and use it to fit a SIREN. The noisy audio decoded from the INR is shown in Fig. 14(b). Then we use our INSP-Net to implicitly process it to a new INR that can be further decoded into denoised audio. It's decoded result is shown in Fig. 14(c). We also provide visualization of the denoising effect in Fig. 14(f).

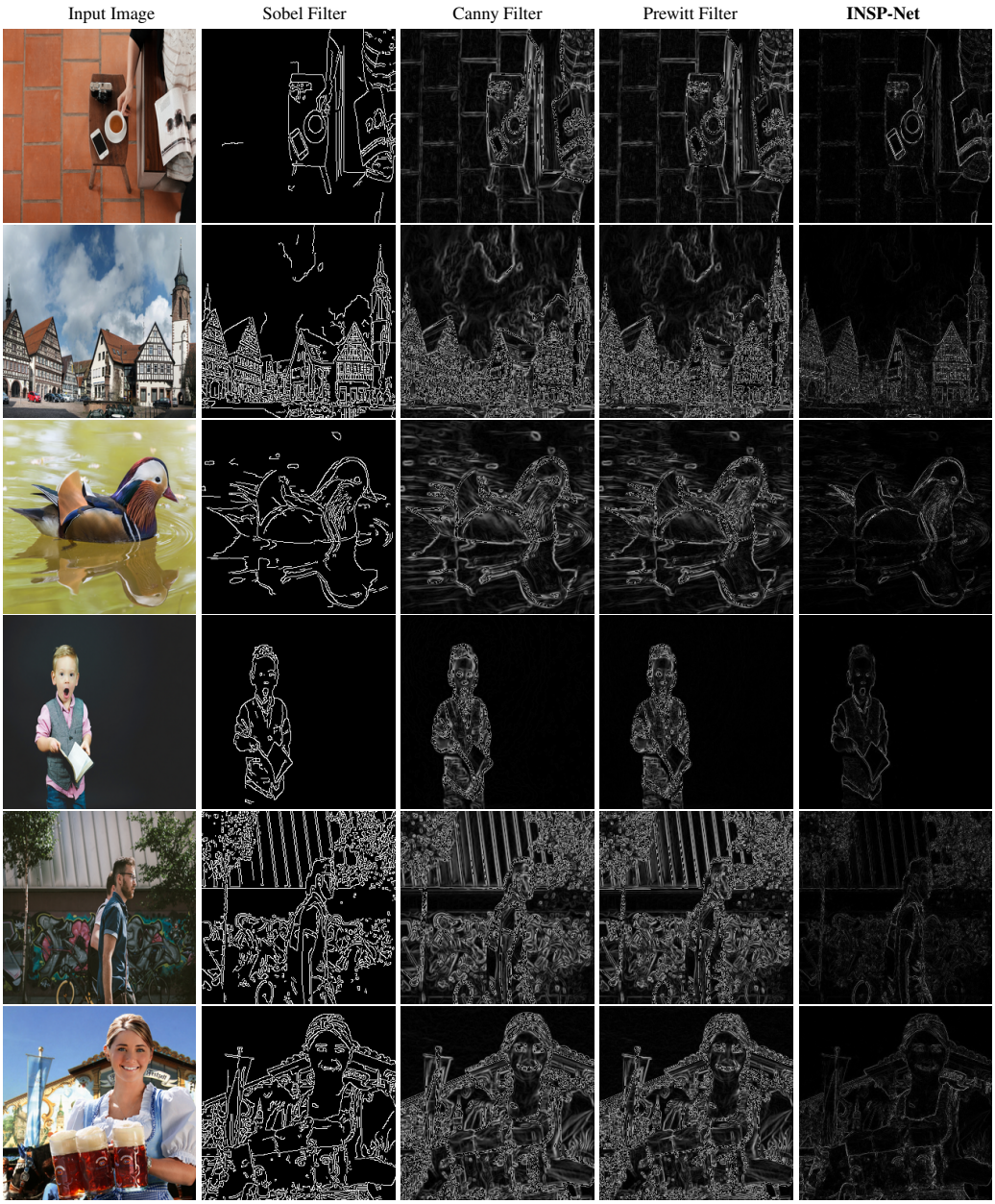


Figure 9: Edge detection. We fit the natural images with SIREN and use our INSP-Net to process implicitly into a new INR that can be decoded into edge maps.

Figure 10: Image denoising. We fit the noisy images with SIREN and train our INSP-Net to process implicitly into a new INR that can be decoded into natural clear images.

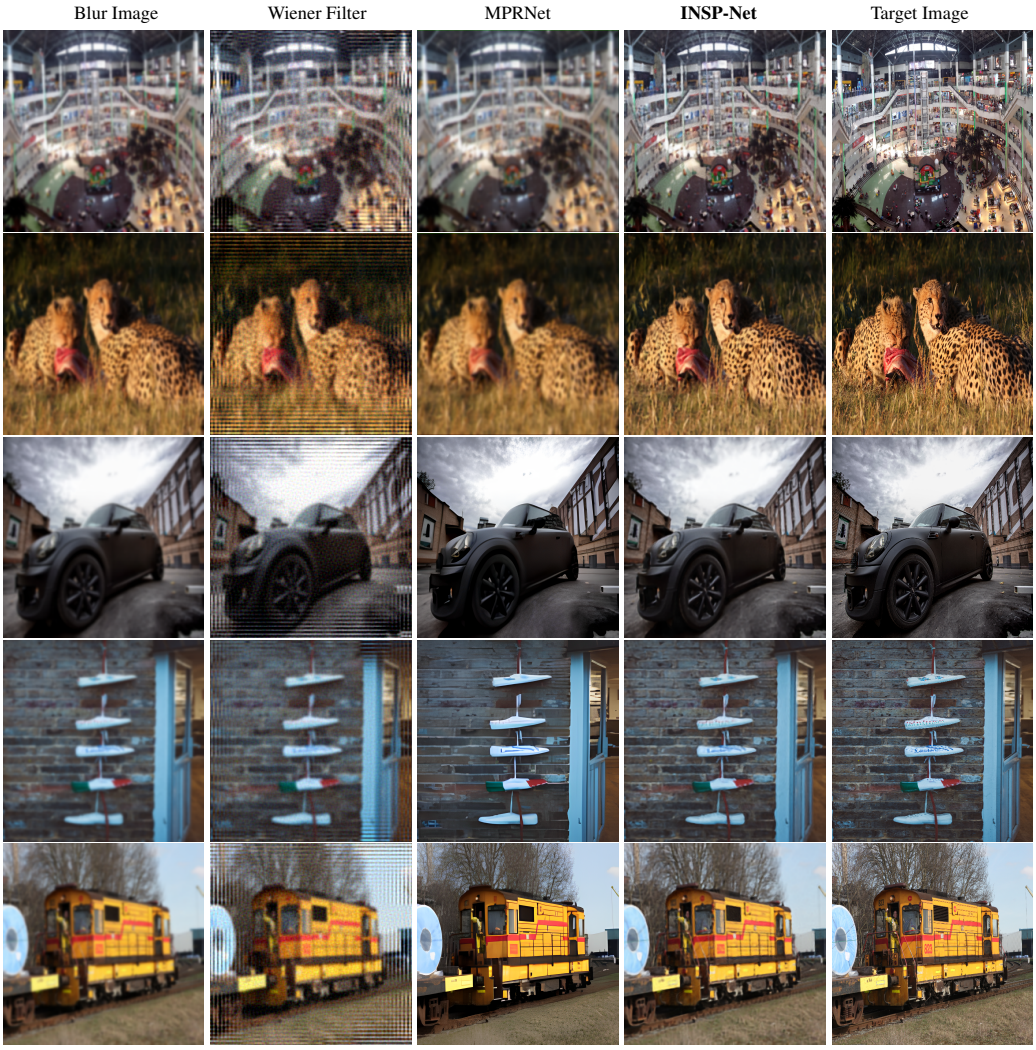


Figure 11: Image deblurring. We fit the blurred images with SIREN and train our INSP-Net to process implicitly into a new INR that can be decoded into clear natural images.

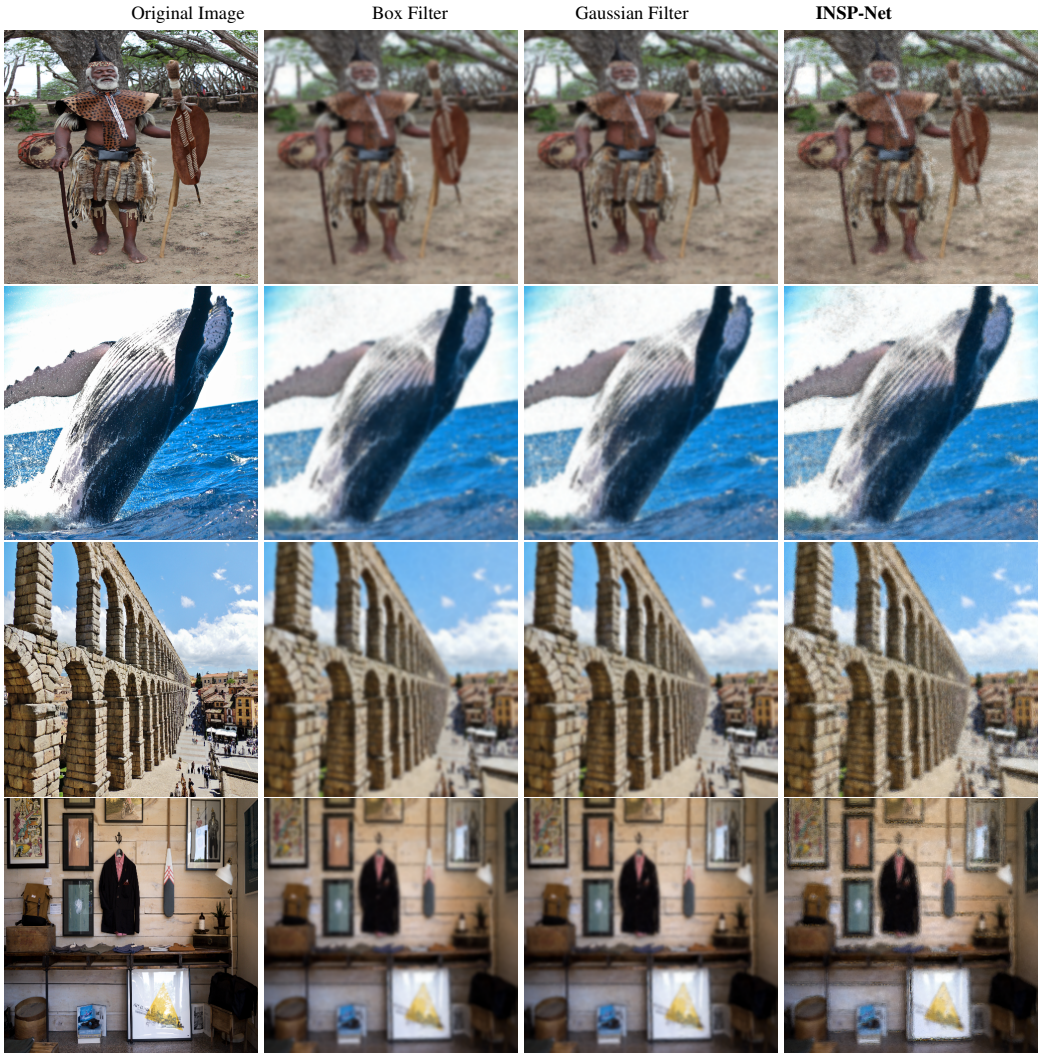


Figure 12: Image blurring. We fit the natural images with SIREN and train our INSP-Net to process implicitly into a new INR that can be decoded into blurred images.

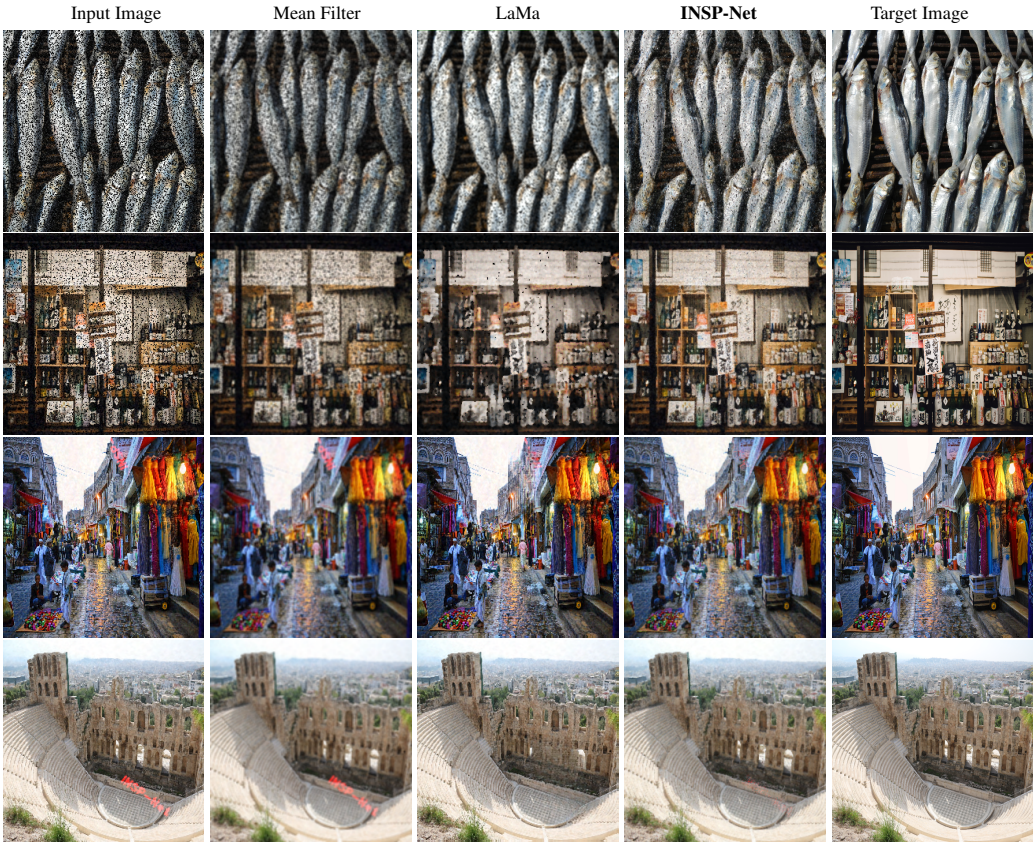
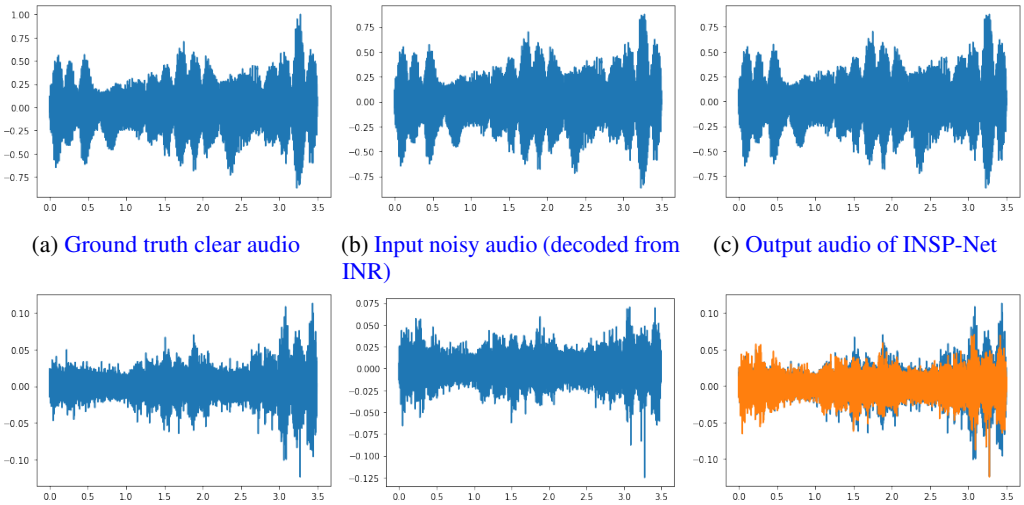


Figure 13: Image inpainting. We fit the input images with SIREN and train our INSP-Net to process implicitly into a new INR that can be decoded into natural images. Note that LaMa requires explicit masks to select the regions for inpainting and the masks are roughly provided. The first two rows contain input images with random pixels erased. The last two rows contain input images with text contamination.



(a) Ground truth clear audio (b) Input noisy audio (decoded from INR) (c) Output audio of INSP-Net

(d) Difference between input and ground truth (e) Difference between output and ground truth (f) blue: (d), orange: (e)

Figure 14: Audio denoising. We fit the noisy audio with SIREN and train our INSP-Net to process implicitly into a new INR that can be decoded into denoised audio.