


# On Architecture Selection for Linear Inverse Problems with Untrained Neural Networks

Yang Sun <sup>1,\*</sup>, Hangdong Zhao <sup>1</sup> and Jonathan Scarlett <sup>1,2,3</sup> 

<sup>1</sup> Department of Computer Science, National University of Singapore, 15 Computing Dr., Singapore 117418, Singapore; dcszhdg@nus.edu.sg (H.Z.); scarlett@comp.nus.edu.sg (J.S.)

<sup>2</sup> Department of Mathematics, National University of Singapore, 10 Lower Kent Ridge Rd., Singapore 119076, Singapore

<sup>3</sup> Institute for Data Science, National University of Singapore, 3 Research Link, Singapore 117602, Singapore

\* Correspondence: yang.sun@comp.nus.edu.sg

**Abstract:** In recent years, neural network based image priors have been shown to be highly effective for linear inverse problems, often significantly outperforming conventional methods that are based on sparsity and related notions. While pre-trained generative models are perhaps the most common, it has additionally been shown that even untrained neural networks can serve as excellent priors in various imaging applications. In this paper, we seek to broaden the applicability and understanding of untrained neural network priors by investigating the interaction between architecture selection, measurement models (e.g., inpainting vs. denoising vs. compressive sensing), and signal types (e.g., smooth vs. erratic). We motivate the problem via statistical learning theory, and provide two practical algorithms for tuning architectural hyperparameters. Using experimental evaluations, we demonstrate that the optimal hyperparameters may vary significantly between tasks and can exhibit large performance gaps when tuned for the wrong task. In addition, we investigate which hyperparameters tend to be more important, and which are robust to deviations from the optimum.

**Keywords:** linear inverse problems; untrained neural networks; compressive sensing; deep decoder; architecture design; hyperparameters



**Citation:** Sun, Y.; Zhao, H.; Scarlett, J. On Architecture Selection for Linear Inverse Problems with Untrained Neural Networks. *Entropy* **2021**, *23*, 1481. <https://doi.org/10.3390/e23111481>

Academic Editor: Gwanggil Jeon

Received: 1 September 2021

Accepted: 6 November 2021

Published: 9 November 2021

**Publisher's Note:** MDPI stays neutral with regard to jurisdictional claims in published maps and institutional affiliations.



**Copyright:** © 2021 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

## 1. Introduction

Linear inverse problems arise in a wide range of application domains, such as computational imaging, optics, and remote sensing. Broadly, the problem consists of measuring a target signal  $\mathbf{x}^* \in \mathbb{R}^n$  via linear measurements of the form

$$\mathbf{y} = \mathbf{A}\mathbf{x}^* + \mathbf{w}, \quad (1)$$

where  $\mathbf{y} \in \mathbb{R}^\ell$  is the dimensionality-reduced observation vector,  $\mathbf{A} \in \mathbb{R}^{\ell \times n}$  is a linear operator that captures the forward process, and  $\mathbf{w} \in \mathbb{R}^\ell$  represents additive noise. The aim is to recover (an estimate of) the unknown signal  $\mathbf{x}^*$  given  $\mathbf{y}$  and  $\mathbf{A}$ . This setup captures a variety of problems including inpainting, denoising, super-resolution, and compressive sensing.

To permit the accurate recovery of  $\mathbf{x}^*$  with limited measurements (possibly  $\ell \ll n$ ), a common approach in recent years has been to adopt explicit mathematical assumptions of low-dimensional structure, such as sparsity in a suitably-defined basis [1]. More recently, it has been observed that *data driven* generative priors can lead to considerable savings in the number of measurements [2], with a typical assumption being that  $\mathbf{x}^*$  can be well-approximated via a pre-trained generative neural network. Such pre-trained models tend to require large amounts of training data, which may be prohibitive in practical applications. In addition, even given large amounts of data, such solutions can suffer from distribution shift, since the training signals may not be fully representative of the test signals.

As an alternative method that can help overcome the above limitations, it has been observed that even *untrained* neural networks can serve as excellent priors for image recovery in linear inverse problems [3,4], where the input to the network is random and the weights are tuned to produce just a single image. In some cases, these techniques are combined with various forms of implicit or explicit regularization, such as early stopping [3], under-parametrized models [4], dropout methods [5], and total variation regularization [6]. Successful applications of the untrained approach have included recovery of natural images [3,4], magnetic resonance imaging [7], X-ray imaging [6], and computed tomography [8].

Broadly speaking, our work is motivated by the following gaps in the literature on inverse problems with untrained neural networks:

- Untrained neural networks invariably come with architectural hyperparameters (e.g., input size, number of layers, convolutional filters used, etc.), and while impressive results have been observed under various architectures, relatively little attention has been paid to how best to select these parameters.
- Untrained neural networks have primarily been applied in the context of recovering images (e.g., natural images, medical images, etc.), but to our knowledge, no detailed study has been given on how the architectural hyperparameters may vary across different signal types (e.g., rough vs. smooth), and different measurement types (e.g., inpainting vs. denoising vs. compressive sensing), nor on the robustness to changing from one setting to another.
- Regarding the signal type, while one-dimensional time-series data has been considered in numerous works on sparsity-based compressive sensing (e.g., neuro-electrical signals [9] and sensor network data [10]), to our knowledge, such signals have received significantly less attention in the context of neural network based priors, with one exception being a one-dimensional Deep Image Prior in [11].

Accordingly, in this paper, focusing on the Deep Decoder approach [4,12], we seek to further explore the utility of untrained neural networks beyond image recovery (in particular, to time-series signals), and more importantly, to better understand the role of architecture selection when recovering signals from different measurement models and signal types. We present algorithms for automatically tuning the architectural hyperparameters, and experimentally observe their behavior in diverse settings of interest. We focus in particular on addressing the following questions:

- (i) To what extent do the optimal hyperparameters vary for different measurement models and signal types?
- (ii) To what extent does the performance degrade when the hyperparameters are tuned for one setting but applied to another?
- (iii) To what extent are the various hyperparameters robust to deviations from their optimal value?

Regarding question (i), we find that the optimal configurations can depend heavily on both the measurement models and the signal types. Accordingly, it is natural in question (ii) that transferring settings can degrade the performance significantly, and we find that this is indeed commonly observed (though not always the case). Finally, regarding question (iii), we identify both examples of robust and non-robust behavior under deviations from the optimal value. These findings are based on experimentation on a diverse range of synthetic and real-world data types, with a particular emphasis on one-dimensional time-series signals.

### 1.1. Related Work

The literature on neural network techniques in inverse problems is rapidly growing [2–4,13–16]. We refer the reader to [17] for a recent survey, and focus here on the most closely-related works.

Two prevailing approaches in the literature are Deep Image Prior [3] and Deep Decoder [4], which adopt a similar high-level approach of tuning network weights to produce a single image. As outlined above, in contrast with approaches based on pre-trained models learned from data [2], Deep Image Prior and Deep Decoder are untrained, and consist of fitting neural network weights to a single image. Follow-up works have also studied different interpretations of Deep Image Prior from the perspective of architecture regularization [18] and Maximum a Posteriori Probability (MAP) estimation [19]. Although concerns have been raised that these approaches may lose information during their intermediate layers [20]; both have shown excellent (and typically comparable) performance in various inverse problems. In many cases, Deep Decoder enjoys the additional benefit of not requiring early stopping due to its relatively simple structure. In this paper, we focus our attention on Deep Decoder, but an analogous investigation of Deep Image Prior may be of interest in future work.

After the introduction of the Deep Decoder model in [4], several follow-up works explored variations and applications, which we outline as follows:

- In [12], several variants of Deep Decoder are introduced depending the presence/absence of upsampling and certain convolution operations. The success of Deep Decoder is primarily attributed to the presence of convolutions with fixed interpolating filters in the neural network architecture. Further details are given in Section 1.2.
- Theoretical guarantees for compressive sensing were given in [14,21]. The former studies the convergence of a projected gradient descent algorithm in underparametrized settings, whereas the latter shows that regular gradient descent is able to recover sufficiently smooth signals even in overparametrized settings.
- Variations of Deep Decoder for medical imaging are given in [7,22], with an additional challenge being combining measurements from multiple coils measuring the same signal in parallel. Additional applications of deep decoder include quantitative phase microscopy [23] and image fusion [20]. In addition, another variant of Deep Decoder for graph signals is given in [24].
- In [25], a method is proposed for combining the benefits of trained and untrained methods, by imposing priors that are a combination of the two.
- In [26], various robustness considerations for neural network based methods (both trained and untrained) are investigated. In particular, (i) both are shown to be sensitive to adversarial perturbations in the measurements; (ii) both may suffer from significant performance degradation under distribution shifts; and (iii) evidence is provided that the overall reconstruction performance is strongly correlated with the ability to recover specific fine details.

The importance of hyperparameter selection was highlighted in [7], but has remained relatively unexplored. The above mentioned more recent work [26] developed independently from ours, and at least one of their observations therein matches one of ours (namely, performance degradation when the hyperparameters are tuned for the wrong data type), but, overall, our work and [7,26] remain mostly separate. Regarding the signal type, in our understanding, the recovery of one-dimensional signals has received much less attention compared to two-dimensional images, with one exception being the application of Deep Image Prior to time series signals in [11].

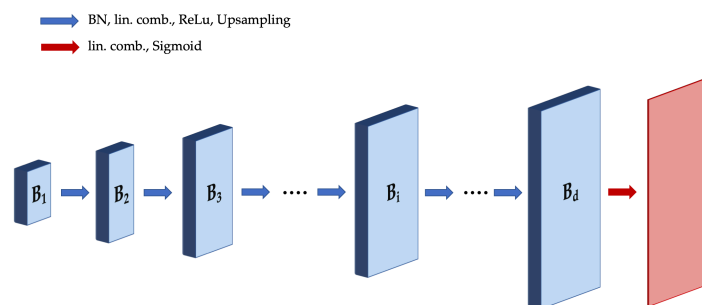
Architecture search is a popular topic in machine learning, particularly in the context of classification problems. Proposed search methods include reinforcement learning [27], parameter sharing [28], and differentiable search [29]; see [30] for a survey. However, to the best of our knowledge, such techniques cannot easily be applied in the context of signal recovery with untrained neural networks, which is notably distinct from the supervised problem of classification. Instead, our approach will use the simpler idea of searching over *parametrized* architectures, building on hyperparameter optimization techniques [31,32]. Having said this, we believe that adapting non-parametrized architecture search methods to our setting could be of significant interest in future work.

### 1.2. Background: Deep Decoder

The Deep Decoder, denoted as  $G(\mathbf{C})$  and parameterized by model weights  $\mathbf{C}$ , transforms a randomly chosen and fixed input tensor  $\mathbf{B}_1 \in \mathbb{R}^{n_1 \times k_1}$ , which is the “latent” input consisting of  $k_1$  many  $n_1$ -dimensional channels, to an  $n_d \times k_{out}$  dimensional image  $\mathbf{x}$ . Here,  $d$  is the depth (i.e., number of layers) of the network. The network transforms the tensor  $\mathbf{B}_1$  to an image using batch normalization [33] (which is equivalent to channel-wise normalization here), upsampling operations, pixel-wise linearly combining of channels, and rectified linear units (ReLUs). In the original version of the Deep Decoder [4], the tensors in the  $(i + 1)$ -th and final layer are given by

$$\begin{aligned} \mathbf{B}_{i+1} &= \text{bn}(\text{relu}(\mathbf{U}_i \mathbf{B}_i \mathbf{C}_i)), \quad i = 1, \dots, d - 1, \\ \mathbf{x} &= \text{sigmoid}(\mathbf{B}_d \mathbf{C}_d), \end{aligned} \tag{2}$$

where the coefficient matrices  $\mathbf{C}_i \in \mathbb{R}^{k_i \times k_{i+1}}$  are the network weights of  $1 \times 1$  convolutions kernels on layer  $i$ ,  $\text{bn}()$  is batch-normalization, and  $\mathbf{U}_i \in \mathbb{R}^{n_{i+1} \times n_i}$  is an upsampling tensor. The model architecture is depicted in Figure 1 in the context of producing 2D images; for 1D signals, the structure is similar with two dimensions “flattened” into one.



**Figure 1.** Illustration of Deep Decoder architecture for 2D signals. Each layer consists of channel-wise batch normalization, pixel-wise linear combining of channels ( $1 \times 1$  convolution), ReLU activation, and upsampling.

Several variants of Deep Decoder were presented in [12], depending on the presence of upsampling and fixed vs. trained convolutional kernels. These models are all closely related, and we focus on a particular one (termed “model (i)” in [12]), which we found to consistently perform best (or equal best) in terms of computation time and recovery performance.

In the model that we focus on, for each layer, an additional fixed-kernel convolution operation is performed before the  $1 \times 1$  convolution described above. Specifically, we have  $\mathbf{U}_d = \mathbf{I}$ , all other  $\mathbf{U}_i$  still as upsampling operators,  $\mathbf{C}_i \in \mathbb{R}^{k_i \times k_{i+1}}$  same as the above, but plus an additional operator  $\mathbf{T}(\mathbf{c}_i)$  performing a convolution with the fixed-kernel  $\mathbf{c}_i \in \mathbb{R}^s$  (where  $s$  is the size of kernel and is a hyperparameter which we will explore in this paper) on layer  $i$ . For instance, in the 1D setting, the following filter of length  $s = 4$  could be used:

$$\mathbf{c}_i = \frac{1}{16} [3 \quad 9 \quad 9 \quad 3]. \tag{3}$$

A filter size of 4 is indeed suggested in [12], but we also allow for other sizes. For a general size  $s$ , we define the center to be  $\frac{s-1}{2}$ , and let the  $i$ -th filter weight be proportional to  $\exp(-2 \text{dist}_i)$ , where  $\text{dist}_i$  is the distance to the center. The filter weights are always normalized to sum to one.

Given the Deep Decoder network  $G(\mathbf{C})$ , the problem of estimating  $\mathbf{x}^*$  given  $(\mathbf{A}, \mathbf{y})$  is performed as follows:

$$\hat{\mathbf{C}} \leftarrow \underset{\mathbf{C}}{\text{minimize}} \|\mathbf{A}G(\mathbf{C}) - \mathbf{y}\|^2, \quad \hat{\mathbf{x}} = G(\hat{\mathbf{C}}), \tag{4}$$

where the “minimize” operation may not necessarily correspond to finding a global minimizer, but rather corresponds to running a variant of Gradient Descent for a certain number of iterations.

Our general goal is for  $\hat{\mathbf{x}} = G(\hat{\mathbf{C}})$  to be a good approximation of  $\mathbf{x}^*$ . In general, we may measure the performance according to some generic loss function  $\ell(\mathbf{x}, \hat{\mathbf{x}})$ . For concreteness, we will focus primarily on the squared loss,  $\ell(\mathbf{x}, \hat{\mathbf{x}}) = \|\mathbf{x} - \hat{\mathbf{x}}\|^2$ , and, in our experiments, we will reparametrize this according to the more widely-adopted Peak Signal-to-Noise Ratio (PSNR):

$$\text{PSNR} = 20 \log_{10} \left( \frac{\text{MAX}_I}{\sqrt{\frac{1}{n} \|\mathbf{x} - \hat{\mathbf{x}}\|^2}} \right), \quad (5)$$

where  $\text{MAX}_I$  is the maximum possible signal value. The PSNR serves as a natural measure for reconstruction accuracy is ubiquitous in the signal processing literature, and has the desirable feature of being invariant to rescaling (e.g., when converting from [0, 1]-valued images to [0, 256]-valued images). On the other hand, our proposed techniques are general and could be used alongside other measures such as the structural similarity index (SSIM).

### 1.3. Hyperparameters and Problem Variables

In view of the above description of Deep Decoder, we focus on optimizing the following important hyperparameters: (i) input tensor size, (ii) number of channels per layer, (iii) number of layers, (iv) filter size for the fixed-kernel, and (v) step size in the Adam optimizer. While further hyperparameters could also be considered (e.g., activation function, other optimization parameters), we found these to consistently work well with fixed choices.

We note that the preceding hyperparameters also implicitly determine the upsampling factor. Specifically, with the output size being fixed according to the problem, further fixing the input size and number of layers also fixes the upsampling factor (assumed to be the same in each layer, up to rounding). In contrast, previous works focused on an upsampling factor of two.

The problem variables that can potentially impact the choice of hyperparameters include the signal type (e.g., slow vs. fast varying), signal length, measurement type (e.g., inpainting, compressive sensing, etc.), compression ratio  $\frac{\ell}{n}$ , noise level, and so on.

## 2. Hyperparameter Selection

In this section, we present two simple and general-purpose algorithms for optimizing architectural parameters, as well as giving some theoretical insight based on statistical learning theory.

We consider a setup in which, for a particular measurement matrix  $\mathbf{A}$ , we have access to a data set  $\mathcal{D} = \{(\mathbf{x}_j, \mathbf{y}_j)\}_{j=1}^m$ , (Our analysis and algorithms also equally apply when  $\mathbf{A}$  is random and the data set takes the form  $\mathcal{D} = \{(\mathbf{x}_j, \mathbf{A}_j, \mathbf{y}_j)\}_{j=1}^m$ .) where each  $\mathbf{y}_j$  is produced from the corresponding  $\mathbf{x}_j$  according to (1). We consider the case that the architecture is parametrized by a list  $H$  of hyperparameters (e.g., input size, number of layers, etc.), and we write  $G(\mathbf{C}) = G_H(\mathbf{C})$  to highlight this dependence. For any specific choice of  $H$ , we can form  $\hat{\mathbf{x}}$  according to (4), and measure the performance according to the loss function  $\ell(\mathbf{x}, \hat{\mathbf{x}})$ . Our goal is to use the training data  $\mathcal{D}$  to find a good choice of  $H$ . To provide insight on this task, we first consider a theoretical viewpoint.

### 2.1. Theoretical Viewpoint

Consider a statistical learning setup, in which the training examples in  $\mathcal{D}$  are assumed to be independently drawn from some unknown distribution  $P_{\mathbf{X}\mathbf{Y}}$  (where  $P_{\mathbf{Y}|\mathbf{X}}$  follows (1) with a fixed measurement matrix  $\mathbf{A}$ , but the noise distribution is unknown). For convenience, we consider the loss as a function of  $(\mathbf{x}, \mathbf{y})$ :

$$\gamma_H(\mathbf{x}, \mathbf{y}) = \ell(\mathbf{x}, \hat{\mathbf{x}}(H, \mathbf{y})), \quad (6)$$

where  $\hat{\mathbf{x}}(H, \mathbf{y})$  is the estimated obtained by running (4) with hyperparameters  $H$  (and with the known measurement matrix  $\mathbf{A}$ ). In this setting, there is a precise notion of the “best” hyperparameter configuration:

$$H^* = \arg \min_{H \in \mathbb{H}} \mathbf{E}[\gamma_H(\mathbf{x}, \mathbf{y})], \quad (7)$$

where  $(\mathbf{x}, \mathbf{y})$  is a fresh sample from  $P_{\mathbf{X}\mathbf{Y}}$ . While  $H^*$  cannot be computed directly (due to  $P_{\mathbf{X}\mathbf{Y}}$  being unknown), we can adopt the widespread idea from statistical learning theory of minimizing the *empirical loss*:

$$\hat{H} = \hat{H}(\mathcal{D}) = \arg \min_{H \in \mathbb{H}} \frac{1}{m} \sum_{j=1}^m \gamma_H(\mathbf{x}_j, \mathbf{y}_j). \quad (8)$$

As well as providing the starting point for practical techniques in subsequent subsections, this empirical loss minimization approach is theoretically principled, giving the following theoretical guarantee analogous to standard PAC-learnability results (e.g., ([34] Section 4.2)).

**Theorem 1.** *If  $H$  takes values within a finite set  $\mathbb{H}$ , and the losses are bounded by (This extends to arbitrary bounds of the form  $\gamma_H(\mathbf{x}, \mathbf{y}) \in [a, b]$  by rescaling.)  $\gamma_H(\mathbf{x}, \mathbf{y}) \in [0, 1]$  for all  $(\mathbf{x}, \mathbf{y})$ , then for a training set  $\mathcal{D}$  of size  $m$  and any  $\eta > 0$ , we have with probability at least  $1 - \eta$  that*

$$\mathbf{E}[\gamma_{\hat{H}}(\mathbf{x}, \mathbf{y})] \leq \mathbf{E}[\gamma_{H^*}(\mathbf{x}, \mathbf{y})] + \sqrt{\frac{2}{m} \left( \log |\mathbb{H}| + \log \frac{2}{\eta} \right)}. \quad (9)$$

The proof follows standard statistical learning theory arguments and can be found in the Appendix A. This result indicates that, to approximate  $H^*$ , it suffices to have a number of training signals growing as  $m = O(\log |\mathbb{H}|)$ . In particular, if there are  $K$  hyperparameters taking a bounded number of values each, this reduces to  $m = O(K)$ , a linear dependence on the number of hyperparameters. However, it is important to note that this is a *worst-case* guarantee, and in Section 3 we will demonstrate the effectiveness of approximately solving (8) even with a very small number of training signals.

In the remainder of the section, we explore practical algorithms for approximately solving (8). We note that, even if each hyperparameter is restricted in advance to take finitely many values, a brute-force evaluation of all configurations is typically prohibitive, since even a single evaluation requires a separate optimization to solve (4). Hence, the goal is to efficiently explore a *subset* of configurations to find an *approximate* solution to (8).

## 2.2. Successive Halving

To approximately solve (8), we first utilize the idea of Successive Halving [31]. The details are given in Algorithm 1, in which the algorithm takes as input the number of configurations  $n$ , a minimum resource level  $r$  and maximum resource level  $R$  (e.g., number of optimization iterations; see below for details), and a reduction factor  $\eta \geq 2$ . The algorithm makes use of the following simple subroutines:

- The function `random_configurations( $n$ )` generates a set of  $n$  random configurations, i.e., for each such configuration, each hyperparameter value is chosen uniformly at random from the pre-specified finite set of possible values.
- The function `evaluate_psnr( $H, r_i$ )` returns the PSNR after running the minimization (4) with the hyperparameter configuration  $H$  and resource level  $r_i$ .
- The function `top_k( $\mathcal{H}, \mathcal{P}, n_i/\eta$ )` finds the  $n_i/\eta$  highest values of PSNR in the list  $\mathcal{P}$ , and returns the corresponding  $n_i/\eta$  configurations in  $\mathcal{H}$ .

Successive Halving uniformly allocates a budget to a set of hyperparameter configurations, evaluates the PSNRs of all configurations, keeps the top  $1/\eta$ , and increases the budget per configuration by a factor of  $\eta$ . This repeats until the maximum per-configuration budget



of  $R$  is reached, and only one configuration remains. The algorithm allocates exponentially more resources to more promising configurations. The resource under consideration could be the number of iterations of stochastic gradient descent, the number of training examples, and the number of random features, etc.; in this paper, we focus on the number of iterations of gradient descent.

---

**Algorithm 1:** Successive Halving for Hyperparameter Optimization.

---

```

input number of configurations  $n$ , maximum iteration index  $s_{\max}$  maximum
resource  $R$ , reduction factor  $\eta$ 
 $\mathcal{H} = \text{random\_configurations}(n)$ 
for  $i \in \{0, \dots, s_{\max}\}$  do
     $n_i = \lfloor n\eta^{-i} \rfloor$ 
     $r_i = \lfloor R\eta^{i-s_{\max}} \rfloor$  % Amount of resource allocated
     $\mathcal{P} = \{\text{evaluate\_psnr}(H, r_i) : H \in \mathcal{H}\}$ 
     $\mathcal{H} = \text{top\_k}(\mathcal{H}, \mathcal{P}, n_i/\eta)$  % Select highest  $n_i/\eta$  PSNR values
end
return the unique hyperparameter configuration  $H$  remaining in  $T$ 

```

---

2.3. Greedy Fine-Tuning

Even after running Successive Halving, it may still be beneficial to perform some fine-tuning operations to obtain a potentially improved hyperparameter configuration. Here, we introduce another simple greedy algorithm to perform such fine-tuning, detailed in Algorithm 2. The algorithm takes as input the baseline configuration  $H$ , and iteratively updates one hyperparameter at a time by choosing whether to slightly increase, slightly decrease, or remain the same. (We do not have any categorical variables in our experiments, but if any were present, they could be handled similarly, e.g., by trying every category or a random subset.) This is continued until a stopping criteria is met (e.g., maximum number of iterations, or every parameter stayed the same).

Let the number of hyperparameters be denoted by  $K$ . The fine-tune levels  $P_1, \dots, P_K$  are user-specified, and may be iteration-dependent; the approach we use will be described in Section 3.2, along with our stopping condition. Once these are selected, the procedure simply iterates through the  $K$  hyperparameters; for each one, we evaluate the PSNR of the two new candidate values (with all other hyperparameters held fixed) in the  $\text{finetune\_single}(H, P_i)$  subroutine and identify the highest PSNR among the three options (the third option being to remain the same), and update accordingly.

---

**Algorithm 2:** Greedy Fine-Tuning for Hyperparameter Optimization.

---

```

input the baseline hyperparameter configuration  $H = \{h_1, \dots, h_K\}$ , where  $h_i$  is
the configuration of the  $i$ -th hyperparameter
while stopping condition not yet met do
    Select potential values  $P_1 = \{h_1^{\text{above}}, h_1^{\text{below}}\}, \dots, P_K = \{h_K^{\text{above}}, h_K^{\text{below}}\}$ 
    for  $i \in \{1, \dots, K\}$  do
         $h_i^* = \text{finetune\_single}(H, P_i)$  % Evaluate PSNRs and choose best
         $H = \text{update}(H, h_i^*)$ 
    end
end
return fine-tuned configuration  $H$ 

```

---

3. Experiments

In this section, we present various experimental findings based on the algorithms proposed on Section 2. We consider a variety of measurement models and signal types described in Section 3.1, as well as further considering natural signals and images in Section 3.6, where we compare with existing baselines. Overall, our experiments are chosen to cover a diverse range of settings representative of those considered in previous

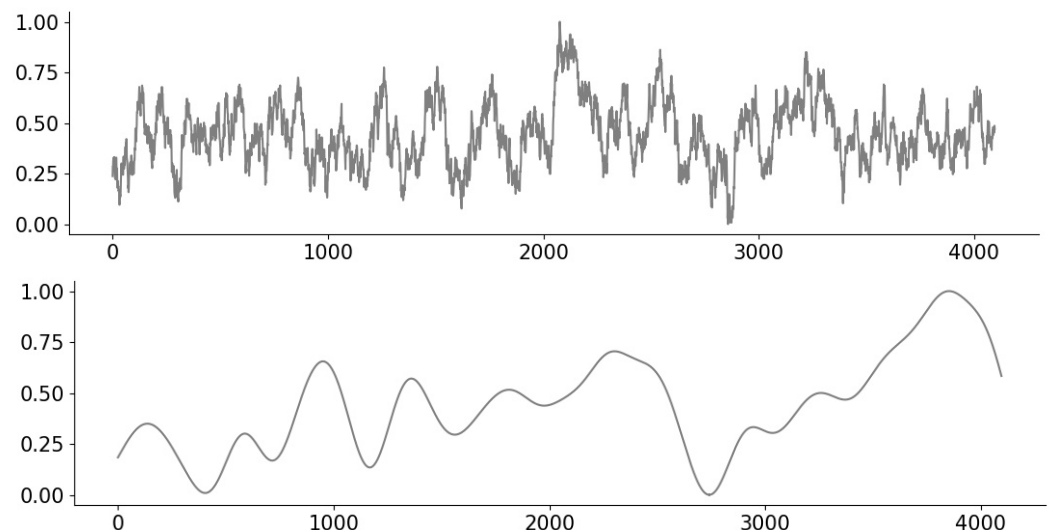
works such as [2,4,11], but we emphasize that our focus is on general-purpose methods, and accordingly, we do not seek to compete against highly specialized techniques on a case-by-case basis. Our code is available at <https://github.com/ethangela/compressed-sensing> (6 October 2021).

### 3.1. Measurement Models and Signals

We consider a variety of measurement models of the general form (1), focusing on the noiseless scenario except where stated otherwise:

- For *random inpainting*, a given fraction of the signal values are masked (i.e., not observed), and we consider the fractions  $9/10$ ,  $3/4$ ,  $1/2$ ,  $1/4$ , and  $1/8$ .
- For *block inpainting*, the signal is divided into blocks (taken to be of length 16 for all 1D signals and of block size  $8 \times 8$  for all 2D signals), and a given fraction of the blocks are masked (i.e., not observed).
- For *compressive sensing*, we consider  $\mathbf{A}$  taking the form of randomly subsampled Gaussian circulant measurements, e.g., see [35]. Such matrices can be viewed as approximating the behavior of i.i.d. Gaussian matrices (e.g., as considered in [2]), but with considerably faster matrix operations. We mostly use 100, 500, or 1024 measurements, but sometimes also consider other values.
- For *denoising*,  $\mathbf{A}$  is the identity matrix (i.e., every entry is observed), but the noise term in (1) corrupts the measurements. We consider i.i.d. Gaussian noise with standard deviations 0.05 and 0.1.

To permit the consideration of distinct signals with a clear notion of lying in a given “class”, we consider synthetic signals drawn from a Gaussian Process (GP) [36] of length  $n = 4096$ . We consider a class of “rough” signals and a class of “smooth” signals on the domain  $[0, 1]$ , with the former being drawn from a GP with an exponential kernel of lengthscale  $\frac{1}{80}$ , and the latter being drawn from a GP with radial basis function (RBF) kernel of lengthscale  $\frac{1}{20}$ . Example signals are shown in Figure 2.



**Figure 2.** Example Gaussian Process (GP) signals corresponding to the exponential kernel (**top**) and RBF kernel (**bottom**).

Whenever we report PSNR values, these are evaluated on *test signals* that are generated separately from the training signals. Specifically, we average over five such test signals to obtain the average PSNR. We also re-generate the random  $\mathbf{A}$  for these test signals, to ensure that we are measuring the generalization to different measurement matrices from the same family.



### 3.2. Algorithmic Details

We apply Successive Halving (Algorithm 1) to optimize the following hyperparameters, initially limiting to the following values:

- #layers  $\in [2, 4, 6, 8, 10, 14, 18, 22, 26, 30]$ .
- #channels  $\in [16, 32, 64, 128, 192, 256, 320, 384, 448, 512]$ .
- input\_size  $\in [2, 4, 8, 16, 24, 32, 48, 128, 256, 512]$ .
- filter\_size  $\in [4, 8, 16, 32, 48, 64]$ .
- step\_size  $\in [0.0005, 0.001, 0.002, 0.003, 0.005, 0.007, 0.01, 0.025, 0.05]$ .

We let the resource in Algorithm 1 be the number of optimization iterations with a maximum value of  $R = 2048$ . The number of possible configurations produced from the five hyperparameter sets above is the cardinality of their Cartesian product, 54,000, and we reduce this to 5400 by taking a random 10% of them. We also set the reduction factor as  $\eta = 3$ , and maximum iteration index as  $s_{\max} = 8$ . The Adam optimizer parameters other than step\_size are kept at their default values.

For greedy fine-tuning (Algorithm 2), we choose set  $p_{\text{above}}$  and  $p_{\text{below}}$  to be of the form  $p_{\text{current}} \pm \Delta$ , where  $\Delta$  is initially chosen to be half the distance to the nearest value in the corresponding list above, and is subsequently halved on each iteration. We stop updating when the distances fall down to 1, except for the non-integer parameter step\_size, for which we use a threshold of 0.0005.

### 3.3. Optimized Parameters for Varying Settings

In Tables 1 and 2, we list the resulting optimized hyperparameters for GPs drawn from the exponential kernel and RBF kernel, respectively. These results provide evidence that the optimal hyperparameters can indeed vary significantly across measurement types and signal types. For instance, we see a pattern that more challenging tasks tend to favor more layers, with the exponential kernel signals (less smooth) tending to choose many more than the RBF kernel signals (more smooth), except random inpainting which is a relatively easier task. On the other hand, the input size tends to be smaller for harder tasks (e.g., block 9/10) and higher for easier tasks (e.g., random 1/8). Our results also suggest that a moderate number of channels (e.g., 150 to 200) and a relatively small step size (e.g., 0.003 to 0.005) tend to perform well across multiple tasks.

Naturally, it is important to not only consider the optimal values, but also the robustness with respect to varying values; this will be done in Section 3.5.

**Table 1.** Optimized hyperparameter configurations (exponential signals).

Measurement	#Channels	#Layers	Input_Size	Filter_Size	Step_Size
block 9/10	110	29	2	11	0.0030
block 3/4	158	25	5	12	0.0030
block 1/2	180	21	18	15	0.0030
block 1/4	180	15	40	30	0.0030
block 1/8	240	9	120	100	0.0070
random 9/10	196	11	45	73	0.0030
random 3/4	158	6	33	73	0.0030
random 1/2	200	4	512	73	0.0030
random 1/4	250	4	512	46	0.0030
random 1/8	250	8	512	2	0.0050
compress 100	83	24	21	11	0.0100
compress 500	146	20	22	9	0.0030
compress 1024	141	29	24	18	0.0030
denoise 0.1	128	10	63	29	0.0020
denoise 0.05	150	4	120	26	0.0070

**Table 2.** Optimized hyperparameter configurations (RBF signals).

Measurement	#Channels	#Layers	Input_Size	Filter_Size	Step_Size
block 9/10	301	3	9	27	0.0055
block 3/4	343	6	14	10	0.0025
block 1/2	227	6	12	37	0.0035
block 1/4	399	2	14	6	0.0100
block 1/8	244	2	38	20	0.0035
random 9/10	112	5	34	27	0.0040
random 3/4	147	7	34	16	0.0045
random 1/2	178	8	42	38	0.0030
random 1/4	166	8	108	26	0.0045
random 1/8	192	10	128	4	0.0025
compress 100	94	7	8	41	0.0080
compress 500	107	10	21	38	0.0050
compress 1024	128	12	19	16	0.0040
denoise 0.1	269	2	13	48	0.0095
denoise 0.05	173	3	15	4	0.0025

### 3.4. Cross-Performance and Transferability

To examine the importance of optimizing for the specific task at hand, we apply the optimized hyperparameter configuration of one type of measurement on other types of measurements, and compare the resulting PSNR values. The results are shown in Tables 3 and 4 for the exponential and RBF signals, with columns indicating training measurement models and rows indicating test measurement models. The diagonals (in which the training and testing are matched to each other) are highlighted via shading.

**Table 3.** Cross-performance PSNR table (exponential signals).

Test \ Train	Block	Block	Random	Random	Compress	Compress	Noise	Noise
	1/2	1/8	1/2	1/8	100	500	0.1	0.05
block 1/2	23.46	22.71	20.21	22.71	23.11	23.23	23.40	21.26
block 1/8	30.73	30.85	27.93	30.85	30.47	30.52	30.51	29.93
random 1/2	34.06	36.11	36.97	36.11	33.44	34.54	33.48	35.00
random 1/8	34.73	37.67	41.61	37.67	33.96	35.62	36.45	35.83
compress 100	17.13	8.56	8.51	8.56	18.98	17.84	15.08	12.78
compress 500	25.87	25.72	17.75	18.56	26.12	25.91	22.11	25.97
denoise 0.1	24.41	21.49	20.02	20.12	25.18	24.51	25.63	24.52
denoise 0.05	29.70	27.61	25.60	26.03	29.83	29.74	28.92	30.14

**Table 4.** Cross-performance PSNR table (RBF signals).

Test \ Train	Block	Block	Random	Random	Compress	Compress	Noise	Noise
	1/2	1/8	1/2	1/8	100	500	0.1	0.05
block 1/2	46.07	43.70	35.94	25.19	39.72	39.17	28.11	45.03
block 1/8	61.33	62.82	62.41	47.80	60.66	59.69	35.28	56.72
random 1/2	67.34	52.59	69.46	65.50	63.34	67.55	29.69	59.88
random 1/8	67.53	53.69	72.25	75.86	63.17	70.12	32.39	58.87
compress 100	39.05	38.87	28.95	18.61	42.94	38.59	30.28	41.86
compress 500	60.52	50.28	54.44	44.42	58.96	61.28	34.94	55.04
denoise 0.1	28.89	34.77	25.73	21.74	32.57	28.31	32.22	34.45
denoise 0.05	36.64	40.26	33.08	29.09	39.26	35.04	27.36	41.23

The fact that the shaded (diagonal) entries have the highest (or very close to highest) value in each row verifies the importance of optimizing for the specific task at hand. In some cases, the gaps can be very significant, e.g., in the ‘compress 100’ row. While the shaded value can sometimes narrowly fall short of being best (e.g., ‘compress 500’ row), the gap is small, and is attributed to (i) the train and test signals still being different, despite

being in the same class, and (ii) the possibility of two different tasks having similar optimal hyperparameters.

### 3.5. Effects of Single Hyperparameters

Next, we seek to examine the importance of single hyperparameters by taking the optimized configurations, and varying one hyperparameter at a time while keeping the rest fixed. While there are too many combinations of measurement models and hyperparameters to show here, representative examples are shown in Figure 3.

We find that a common feature in these plots is an “arch” shape, where the peak indicates the optimal value, though this is not always the case (e.g., for other signals/measurement models, *filter\_size* was significantly more flat). A flat PSNR curve indicates robustness with respect to varying its value, whereas a highly varying PSNR curve indicates that the hyperparameter is particularly important to optimize. For instance, the former scenario is observed in sub-figures (a,h).

We additionally comment that these curves can help identify the impact of over-estimating vs. under-estimating the optimal parameter choices for certain parameters. For example, sub-figures (d,g) indicate that decreasing the input size from its optimal value can degrade the performance much more compared to when increasing the value. Hence, if one were faced with uncertainty about exactly where the optimal value lies (e.g., due to possible future changes in the signal distribution or measurement type), one may prefer to favor a slightly higher input size in this example.

### 3.6. Real-World Data and Comparisons to Baselines

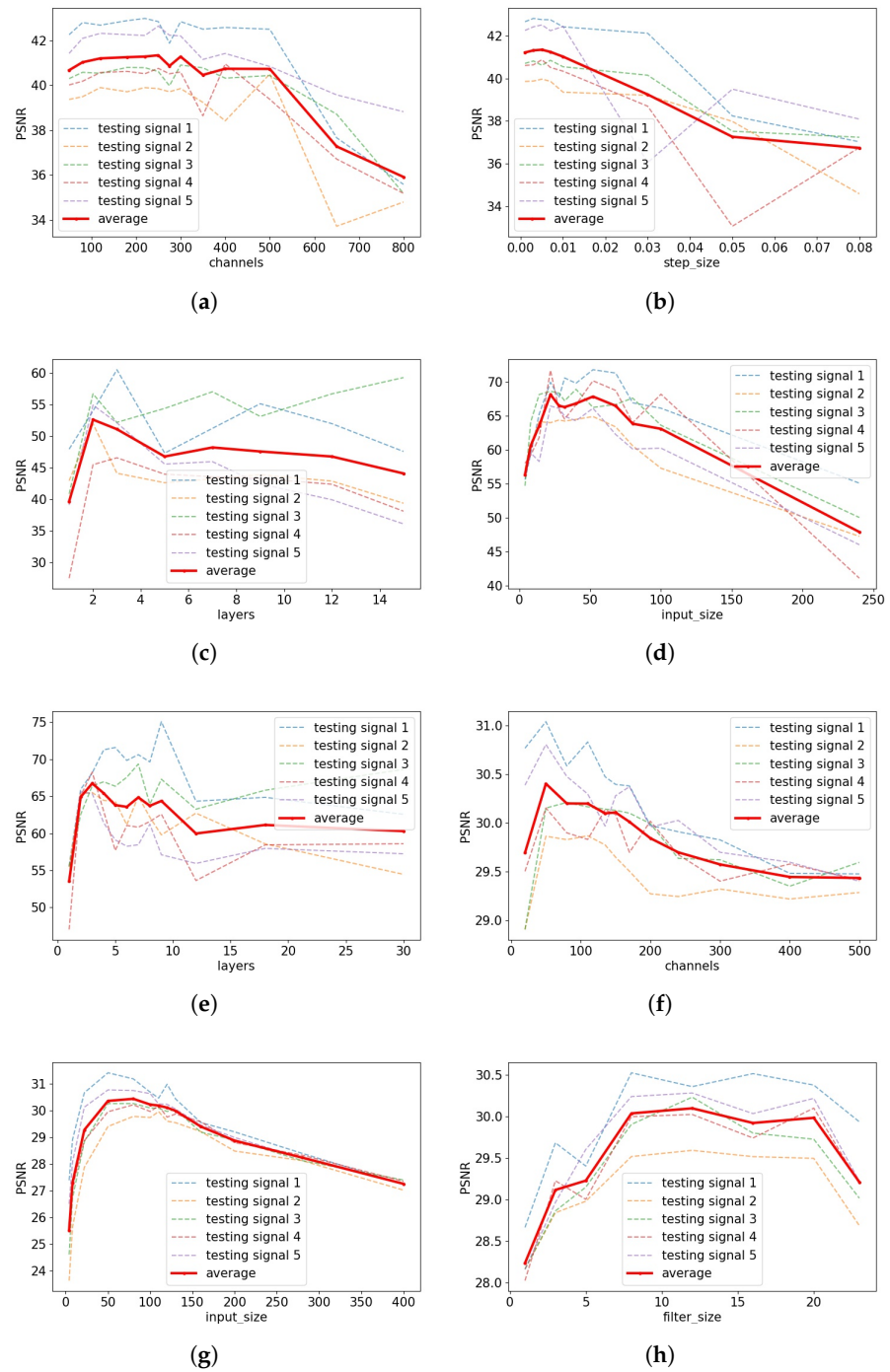
Here, we compare the optimized Deep Decoder to a variety of baselines, as well as moving from synthetic signals to real-world data.

For 1D real time-series data, we use hourly recordings of NO<sub>2</sub> and O<sub>3</sub> levels in air quality of an Italian town throughout 2004–2005 [37]. We split the recordings into signals of length  $n = 1024$  for O<sub>3</sub> and  $n = 512$  for NO<sub>2</sub>, and normalize the range to  $[0, 1]$ . As a data-cleaning pre-processing step, we trim the original signal to remove missing values, while “stitching together” the pieces in a manner that maintains continuity. Examples of the signals are shown in Figure 4.

For 2D image signals, we use the CelebA dataset [38] cropped to size  $128 \times 128 \times 3$ . We note that, when considering CelebA data, the model structure is suitably adapted for 2D images. For example, the filter of fixed-kernel convolution introduced in Equation (3) is expanded from a vector to a square matrix, and similarly for the upsampling operations.

Similar to our synthetic experiments, we maintain a train/test separation for our real-world data experiments, optimizing the hyperparameters using the training signals but evaluating the performance on a *separate* set of test signals. For the 1D air quality data, the training size is 3 and the test size is 5, while for CelebA the training size is 5 and test size is 8.

We select four baseline models to compare against: Original Deep Decoder (Org. DD), original Deep Image Prior (Org. DIP), Lasso with the wavelet basis (LassoW), and Total Variation regularization (TV Norm). The first two methods are the pioneering untrained neural network priors discussed in Section 1 with fixed ‘default’ parameters, and the latter two methods are widely-used conventional priors. The hyperparameter configuration of Org. DD [12] is: *#channels* = 320 for inpainting and *#channels* = 128 for the other tasks, *#layers* = 6, *input\_size* = 128, *step\_size* = 0.01, and *filter\_size* = 4. The hyperparameters of Org. DIP are sourced from the deep generative CNN architecture (U-Net) embedded in Org. DIP. We follow [3] and set the number of up/down layers as 5, the number of input channels as 32, and the number of channels for tensors in middle layers as 128. The number of optimization iterations for both Org. DD and Org. DIP is fixed to 10,000.



**Figure 3.** Example single-hyperparameter plots for various signals in the tasks of random inpainting (random 1/8), block inpainting (block 1/4), denoising (denoise 0.05), and compressive sensing (compress 500). **(a)** #channels, inpainting, exponential; **(b)** step\_size, inpainting, exponential; **(c)** #layers, inpainting, RBF; **(d)** input\_size, compressing, RBF; **(e)** #layers, compress, RBF; **(f)** #channels, denoising, exponential; **(g)** input\_size, denoising, exponential; **(h)** filter\_size, denoising, exponential.

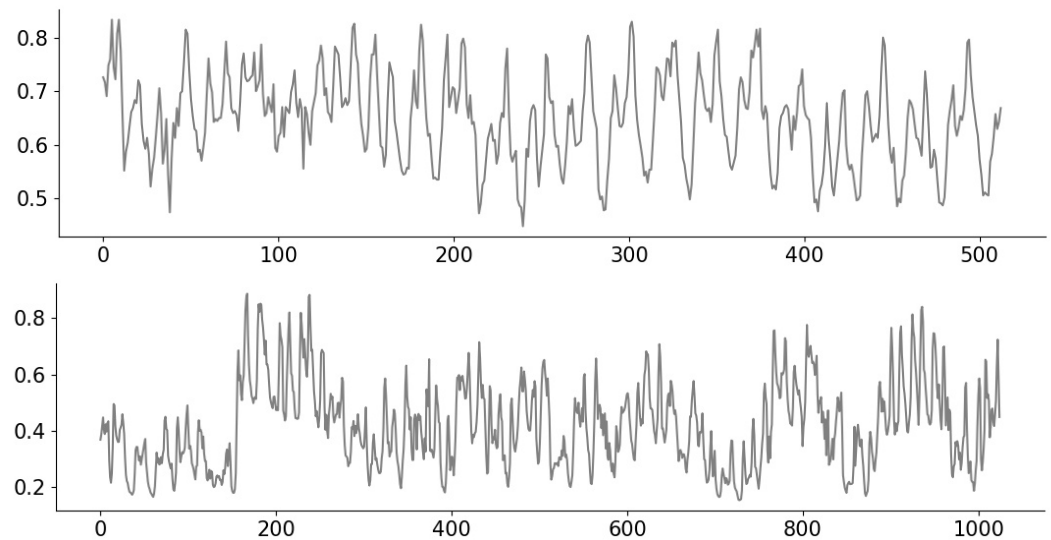


Figure 4. Examples of air quality time series signals for NO2 (top) and O3 (bottom).

For LassoW and TV Norm, we optimize the hyperparameter for each type of signal and measurement, respectively. This hyperparameter, typically denoted  $\lambda$ , controls the weighting of the penalty to the loss function. The maximum number of optimization iterations for both LassoW and TV Norm is set as 2000.

Tables 5 and 6 give the average PSNR for each estimator with selected measurements for GP signals, and Tables 7–9 give the results for NO2 data, O3 data, and CelebA images, respectively. We see that, throughout these experiments, Opt. DD is consistently either the best performing or very close to being so. The gains can be particularly significant for challenging tasks (e.g., compress 100 in Table 5 and compress 25 in Tables 7 and 8), and for tasks with aspects most different from those that the existing methods were optimized for (e.g., DIP is less suited to RBF signals in Table 6).

Table 5. PSNR comparisons among baseline models (exponential signals).

Est. \ Mea.	Opt. DD	Org. DD	Org. DIP	TV Norm	LassoW
block 1/2	<b>22.94</b>	20.99	21.45	22.67	17.72
random 3/4	<b>28.07</b>	27.74	25.85	26.67	17.49
compress 100	<b>18.16</b>	11.44	16.07	16.59	5.69
dno 0.05	<b>29.72</b>	26.81	28.32	29.44	26.86

Table 6. PSNR comparisons among baseline models on (RBF signals).

Est. \ Mea.	Opt. DD	Org. DD	Org. DIP	TV Norm	LassoW
block 1/2	<b>49.06</b>	43.39	29.36	32.58	18.25
random 3/4	<b>65.03</b>	64.69	50.04	42.78	19.47
compress 100	<b>43.32</b>	42.78	17.48	21.21	6.20
dno 0.05	<b>40.77</b>	36.76	29.03	40.95	27.36

**Table 7.** PSNR comparisons among baseline models (NO2 signals).

Mea.	Est.	Opt. DD	Org. DD	Org. DIP	TV Norm	LassoW
block 3/4		<b>22.08</b>	16.60	16.94	18.81	16.72
block 1/2		<b>21.51</b>	19.95	19.57	19.04	19.19
block 1/8		<b>27.45</b>	26.25	27.71	24.43	21.59
random 3/4		<b>24.81</b>	24.73	24.81	20.73	17.39
random 1/2		<b>30.00</b>	29.82	27.93	23.01	18.67
random 1/8		<b>40.73</b>	37.90	34.81	27.22	24.47
compress 25		<b>22.61</b>	13.36	10.92	11.99	5.42
compress 256		<b>31.00</b>	30.90	20.97	18.48	16.89
dno 0.1		24.66	20.53	22.64	<b>25.71</b>	21.41
dno 0.05		27.58	26.77	28.24	<b>30.02</b>	22.55

**Table 8.** PSNR comparisons among baseline models (O3 signals).

Mea.	Est.	Opt. DD	Org. DD	Org. DIP	TV Norm	LassoW
block 3/4		16.00	15.22	15.95	<b>17.58</b>	15.71
block 1/2		<b>18.76</b>	16.92	18.13	18.64	17.65
block 1/8		<b>25.27</b>	24.91	24.23	21.79	21.53
random 3/4		<b>21.71</b>	21.45	21.10	20.95	16.33
random 1/2		<b>27.66</b>	27.23	24.49	25.19	18.73
random 1/8		<b>38.05</b>	34.43	31.73	33.71	23.65
compress 25		<b>16.70</b>	11.90	8.10	14.72	6.60
compress 256		<b>22.88</b>	22.42	21.80	20.34	15.21
dno 0.1		20.43	22.55	22.45	<b>23.41</b>	20.46
dno 0.05		27.11	<b>28.05</b>	27.63	27.04	20.88

**Table 9.** PSNR comparisons among baseline models (CelebA images).

Mea.	Est.	Opt. DD	Org. DD	Org. DIP	TV Norm	LassoW
block 3/4		<b>18.70</b>	16.66	16.23	16.03	12.80
block 1/2		<b>23.65</b>	22.67	20.82	18.03	15.68
block 1/4		<b>27.77</b>	27.43	25.42	22.21	17.35
random 3/4		<b>30.07</b>	28.15	24.59	19.93	16.63
random 1/2		<b>35.65</b>	32.21	29.84	25.98	16.63
random 1/4		<b>39.82</b>	36.63	32.44	31.21	18.09
compress 400		<b>20.11</b>	17.59	18.27	9.35	6.48
compress 4096		<b>27.87</b>	26.06	23.77	14.89	12.57
dno 0.1		<b>38.45</b>	38.22	31.15	24.07	18.47
dno 0.05		<b>49.12</b>	38.01	31.15	28.27	19.77

The only case in which Opt. DD appears to underperform is denoising on 1D time-series data, in which the TV norm approach works unusually well (in comparison to its worse performance in many other cases). In one of these denoising entries, Opt. DD is in fact slightly below Org. DD, and we believe that this is due to the fact that having a small training set size can sometimes limit the generalization performance (e.g., if a test signal has some distinct features). However, among our diverse set of experiments, we otherwise found no such cases.

To complement the numerical values, we give some examples of reconstructed signal segments in Figures 5–8. We observe that, at least in these examples, Opt. DD is able to better match the true signal with fewer artifacts. We also visualize the comparisons of reconstructed CelebA images among all baselines in Figures 9 and 10, and we believe that they are consistently the most visually similar to the originals, thereby matching the consistent PSNR improvements observed in Table 9.



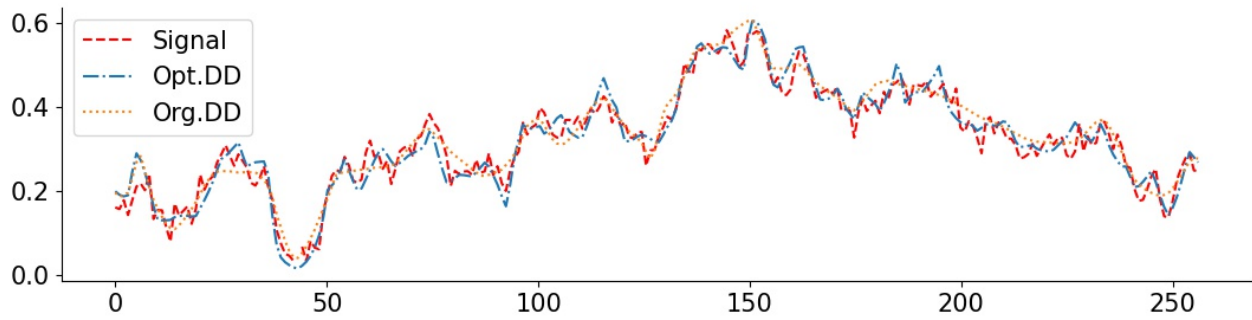


Figure 5. Exponential signal with denoising (0.05): PSNR(Opt. DD) = 29.79, PSNR(Org. DD) = 23.18.

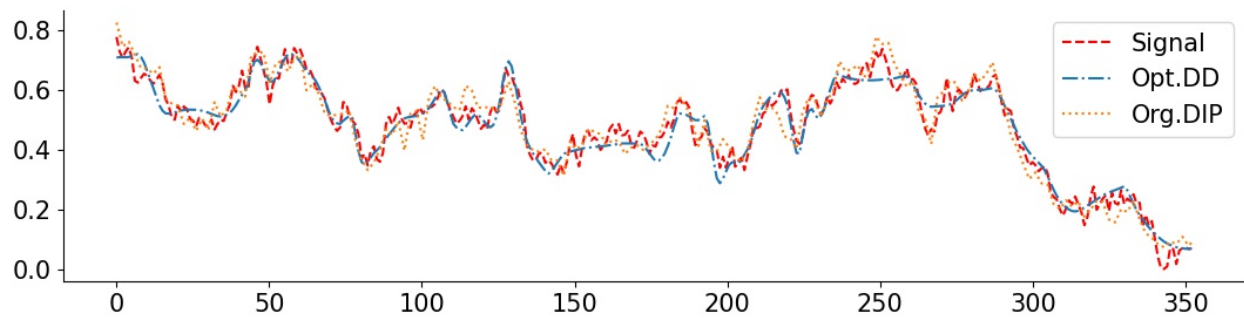


Figure 6. Exponential signal with compressing (1024): PSNR(Opt. DD) = 27.91, PSNR(Org. DIP) = 27.01.

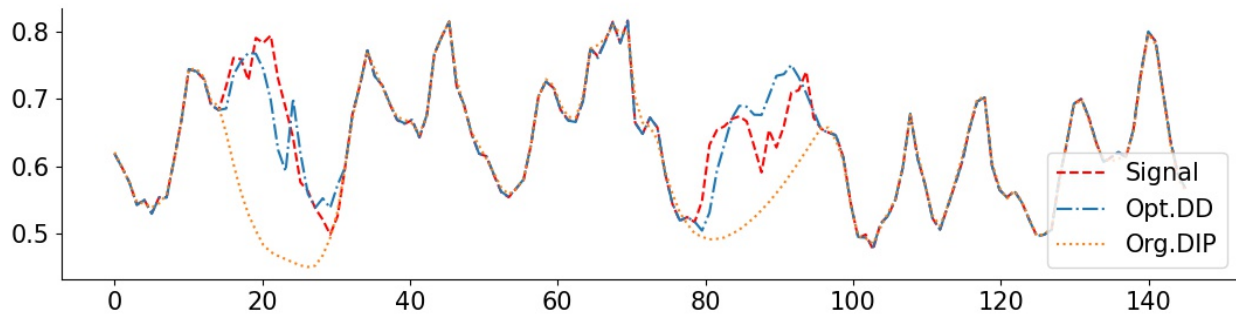


Figure 7. NO<sub>2</sub> signal with inpainting (block 1/4): PSNR(Opt. DD) = 34.16, PSNR(Org. DIP) = 33.45. The large deviations of the Org. DIP curve correspond to locations of missing blocks.

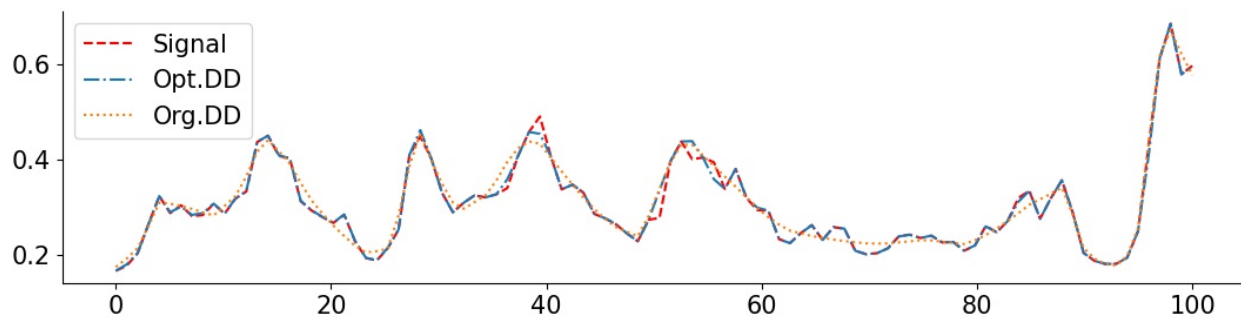


Figure 8. O<sub>3</sub> signal with inpainting (random 1/8): PSNR(Opt. DD) = 37.26, PSNR(Org. DD) = 34.31.

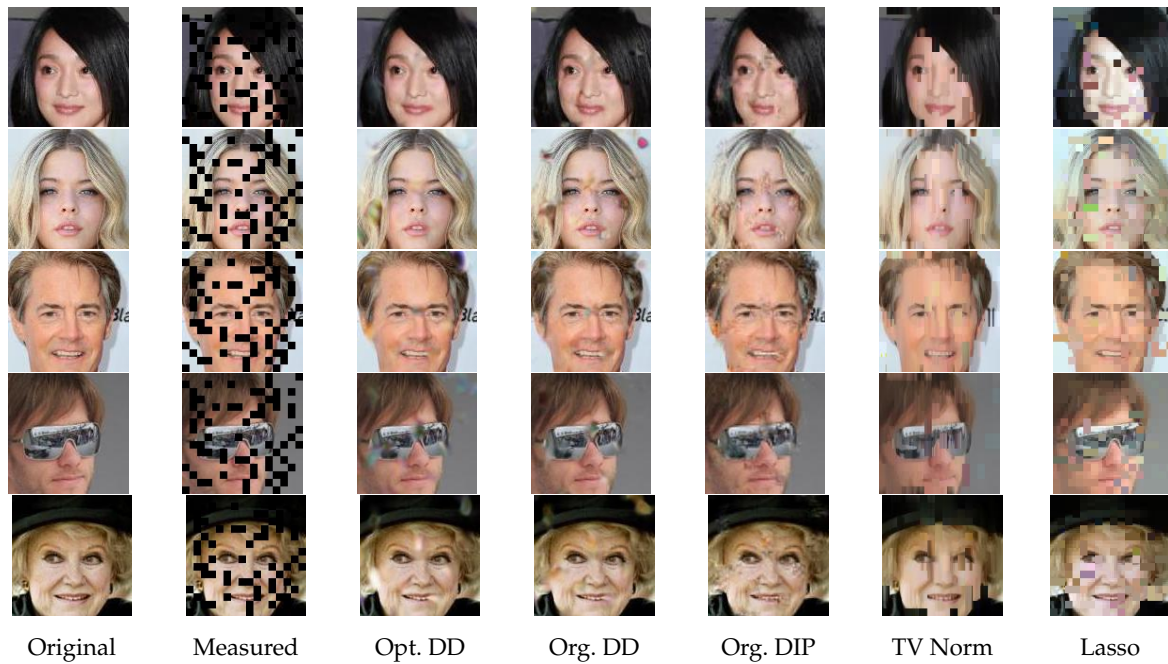


Figure 9. CelebA images with inpainting (block 1/4).

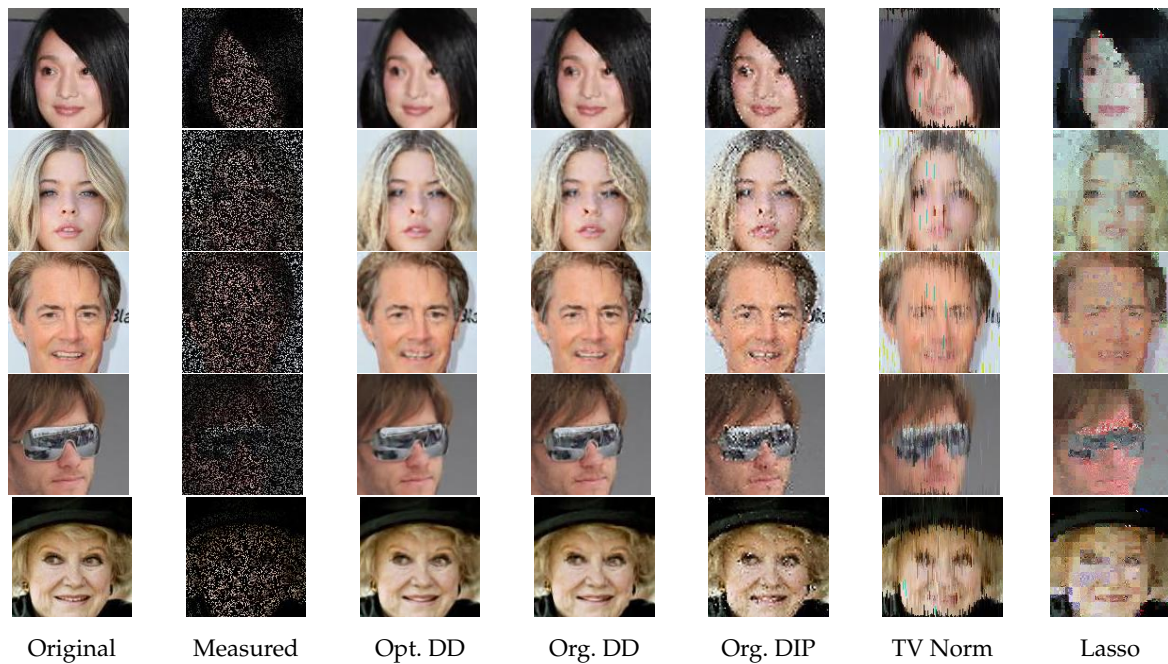


Figure 10. CelebA images with inpainting (random 3/4).

### 3.7. Accelerated Multi-Coil MRI Data

In this final experiment, we apply our proposed algorithms on data from Magnetic Resonance Imaging (MRI). MRI is a widely-used medical imaging technique that promises impressive accuracy, but is prone to requiring long scans that may cause discomfort. Starting with the pioneering work of [39], various compressive sensing based approaches have been proposed for taking fewer measurements and reducing the scan time.

In this experiment, we switch to a slightly more general observation model and consider a variant of the Deep Decoder specifically targeted at this application; we proceed by giving the relevant details. The main difference from the measurement model (1) is

that measurements are taken from *multiple sensors* (coils). In addition, MRI machines are invariably constrained to take measurements in the Fourier domain. Thus, the goal is to recover an image  $\mathbf{x}^* \in \mathbb{C}^n$  from a set of measurements of the form

$$\mathbf{y}_i = \mathbf{M}\mathbf{F}\mathbf{x}^* + \mathbf{w}, \quad i = 1, \dots, n_c, \quad (10)$$

where  $\mathbf{M} \in \mathbb{R}^{\ell \times n}$  is a mask that indicates which Fourier coefficients are observed, (Specifically, each row of  $\mathbf{M}$  has a single 1 at the observed coefficient, and the remaining entries are zero.)  $\mathbf{F} \in \mathbb{R}^{n \times n}$  is the Fourier transform operation,  $n_c$  is the number of magnetic coils, and  $\mathbf{w} \in \mathbb{R}^\ell$  still represents additive noise. Each  $\mathbf{y}_i \in \mathbb{R}^\ell$  represents the  $\ell$  measurements from a single coil, with the idea being that combining measurements from multiple coils improves accuracy. In general, each coil's measurements may further be weighted according to a *sensitivity map* (i.e., some regions appear brighter than others), but this is omitted in (10) since we only seek to follow an experimental setup from [7] that similarly omitted sensitivity maps. The loss function in (4) is updated as follows:

$$\hat{\mathbf{C}} \leftarrow \underset{\mathbf{C}}{\text{minimize}} \frac{1}{2} \sum_{i=1}^{n_c} \|\mathbf{M}\mathbf{F}\mathbf{G}(\mathbf{C}) - \mathbf{y}_i\|^2. \quad (11)$$

We work with the recently-released fastMRI dataset [40] which consists of train/validation set of fully-sampled measurements of knees taken with  $n_c = 15$  coils. We replicate an experiment from [7] to test the performance of three different generators, Deep Image Prior, Deep Decoder, and Deep Decoder variant specifically called ConvDecoder which is specifically targeted towards the MRI application. Specifically, the difference is that, while Deep Decoder uses bi-linear upsampling and  $1 \times 1$  convolutions, the ConvDecoder uses Nearest-Neighbor up-sampling and a  $3 \times 3$  convolutional layer. We replace the constant 3 here by a tunable parameter `filter_size`.

In [7], Deep Decoder and ConvDecoder are tuned through a basic grid search, and our objective here is to tune these networks through our proposed algorithms and compare the results. The relevant additional details of this experiment are as follows:

- The mask  $\mathbf{M}$  is chosen to be a standard 1D variable-density mask (i.e., random or equi-spaced vertical lines across the Fourier space), and is randomly chosen for each run. We do not add  $\mathbf{w}$  explicitly, as the ground truth images already contain some noise.
- Following [7], three additional image comparison metrics are considered along with the PSNR, namely, the Visual Information Fidelity (VIF) [41], Structural Similarity Index (SSIM) [42], and Multi-Scale SSIM (MS-SSIM) [43]. However, following our previous sections, all hyperparameter tuning is done with respect to the PSNR.
- We optimize the same hyperparameters for ConvDecoder as our previous experiments, but the possible settings of `filter_size` for ConvDecoder during Successive Halving are adjusted as `filter_size`  $\in$  [2,3,4,5,6,7], in view of all considered filter sizes being small in [7]. For consistency with [7], we also change the number of optimization iterations to 20,000 for all methods.
- Following [7], we do not consider `filter_size` for Deep Decoder, and instead use the original Deep Decoder as introduced in (2), rather than the variant with an additional fixed-kernel convolution introduced just above (3).
- Each scan of a knee from fastMRI consists of a number of slices, each of which is a 2D image, and together the images form a 3D volume. We choose the the middle slice of the volume to obtain each image, and discard the other slices. The train and test sizes are set as 4 and 16, respectively.

Table 10 presents the evaluation performance of the optimized DD and ConvDD methods vs. the original variants and the original Deep Image Prior, where “original” means making use of the grid search hyperparameters from [7]. We observe that both Opt. ConvDD and Opt. DD are able to slightly improve on the original versions, again indicating the utility and versatility of our tuning algorithms. A minor exception is the

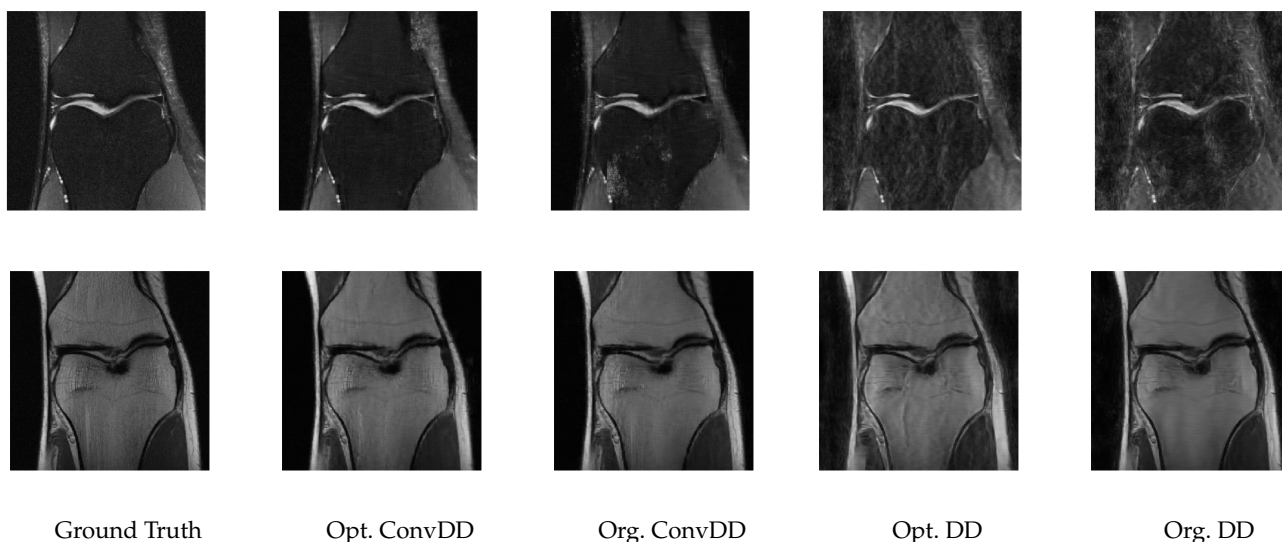


SSIM, but the difference is very marginal; perhaps most important is the PSNR, since this is the metric that was optimized.

**Table 10.** Comparisons of various performance metrics and hyperparameter configurations (fastMRI images). The variant of Deep Decoder used here does not have a filtering step, so filter\_size is omitted.

Method	VIF	MS-SSIM	SSIM	PSNR	#Channels	#Layers	Input_Size	Filter_Size	Step_Size
Opt. ConvDD	<b>0.9674</b>	<b>0.9429</b>	0.8212	<b>31.8063</b>	252	8	5	3	0.002
Org. ConvDD	0.9599	0.9422	<b>0.8259</b>	31.0642	256	8	4	3	0.008
Opt. DD	0.6021	0.8204	0.6515	28.2947	352	9	18	-	0.002
Org. DD	0.5725	0.8029	0.6529	28.2217	368	10	16	-	0.008
Org. DIP	0.5644	0.8644	0.5163	26.9310	256	16	(640,368)	3	0.008

Two sets of sample reconstructions are shown in Figure 11. These reconstructions highlight that the visual quality can be improved in some cases (top row) but remains relatively unchanged in other cases (bottom row). The details of the hyperparameter configurations are also shown in the right part of Table 10. The most notable hyperparameter change from our tuning algorithm was the optimized step size, with a default value of 0.008 but our optimized value given by the smaller value of 0.002.



**Figure 11.** Sample reconstructions of multi-coil knee measurements from fastMRI images.

#### 4. Conclusions

We have studied the role of architecture selection in signal recovery via untrained neural networks, with some of the main implications including (i) different measurement models and signals may benefit from significantly different hyperparameters; (ii) the performance may drop significantly when transferring configurations directly from one setting to another, and (iii) certain hyperparameters tend to exhibit better robustness to deviations from the optimum than others.

Possible directions for future work include performing a similar study of hyperparameter selection for Deep Image Prior [3], and perhaps more ambitiously, exploring methods that search *directly over architectures* rather than only hyperparameters of a pre-specified class of architectures (e.g., see [29] and the references therein in the context of neural networks for classification).

**Author Contributions:** Y.S. conceptualized the problem, performed and analyzed the experiments, and led the writing. H.Z. assisted with coding and the generation of the experimental results.

J.S. provided ongoing supervision and corrections, and assistance with the writing. All authors have read and agreed to the published version of the manuscript.

**Funding:** This research was funded by the Singapore National Research Foundation (NRF) under Grant No. R-252-000-A74-281.

**Data Availability Statement:** This research uses four data sets: (i) Synthetic generated data with code available at <https://github.com/ethangela/compressd-sensing> (accessed 6 October 2021); (ii) Air quality dataset from <http://archive.ics.uci.edu/ml/datasets/Air+Quality> (accessed 6 October 2021); (iii) CelebA dataset from <https://mmlab.ie.cuhk.edu.hk/projects/CelebA.html> (accessed 6 October 2021); (iv) fastMRI dataset from <https://fastmri.org/dataset/> (accessed 6 October 2021).

**Conflicts of Interest:** The authors declare no conflict of interest.

### Appendix A. Proof of Theorem 1

We first define the empirical average

$$\hat{\mathbf{E}}[\gamma_H(\mathbf{x}, \mathbf{y})] := \frac{1}{m} \sum_{j=1}^m \gamma_H(\mathbf{x}_j, \mathbf{y}_j), \tag{A1}$$

which is a random variable depending on the training data  $(\mathbf{x}_1, \mathbf{y}_1), \dots, (\mathbf{x}_m, \mathbf{y}_m)$ . In addition, to reduce notation, we define

$$\Delta_m = \mathbf{E}[\gamma_{\hat{H}}(\mathbf{x}, \mathbf{y})] - \mathbf{E}[\gamma_{H^*}(\mathbf{x}, \mathbf{y})]. \tag{A2}$$

We expand this definition as

$$\begin{aligned} \Delta_m &= (\mathbf{E}[\gamma_{\hat{H}}(\mathbf{x}, \mathbf{y})] - \hat{\mathbf{E}}[\gamma_{\hat{H}}(\mathbf{x}, \mathbf{y})]) + (\hat{\mathbf{E}}[\gamma_{\hat{H}}(\mathbf{x}, \mathbf{y})] - \hat{\mathbf{E}}[\gamma_{H^*}(\mathbf{x}, \mathbf{y})]) \\ &\quad + (\hat{\mathbf{E}}[\gamma_{H^*}(\mathbf{x}, \mathbf{y})] - \mathbf{E}[\gamma_{H^*}(\mathbf{x}, \mathbf{y})]) \tag{A3} \\ &\leq \left| \hat{\mathbf{E}}[\gamma_{\hat{H}}(\mathbf{x}, \mathbf{y})] - \mathbf{E}[\gamma_{\hat{H}}(\mathbf{x}, \mathbf{y})] \right| + \left| \hat{\mathbf{E}}[\gamma_{H^*}(\mathbf{x}, \mathbf{y})] - \mathbf{E}[\gamma_{H^*}(\mathbf{x}, \mathbf{y})] \right|, \end{aligned}$$

where the last line holds because  $\hat{\mathbf{E}}[\gamma_{\hat{H}}(\mathbf{x}, \mathbf{y})] \leq \hat{\mathbf{E}}[\gamma_{H^*}(\mathbf{x}, \mathbf{y})]$  by the definition of  $\hat{H}$  in (8).

Since the loss function is assumed to be bounded within  $[0, 1]$ , and both the  $(\mathbf{x}, \mathbf{y})$  pairs and training signals  $\{\mathbf{x}_j, \mathbf{y}_j\}_{j=1}^m$  are i.i.d. and sampled from the same distribution  $P_{\mathbf{X}\mathbf{Y}}$ , we may apply Hoeffding’s inequality ([44] Section 2.6) to obtain

$$\mathbf{P}\left(\left|\hat{\mathbf{E}}[\gamma_H(\mathbf{x}, \mathbf{y})] - \mathbf{E}[\gamma_H(\mathbf{x}, \mathbf{y})]\right| > \epsilon_0\right) \leq 2 \exp(-2m\epsilon_0^2) \tag{A4}$$

for any fixed  $H$  and constant  $\epsilon_0 > 0$ . Furthermore, as we are considering finite  $\mathbb{H}$ , we can simply apply the union bound to (A4) to obtain

$$\mathbf{P}\left(\bigcup_{H \in \mathbb{H}} \left\{ \left| \hat{\mathbf{E}}[\gamma_H(\mathbf{x}, \mathbf{y})] - \mathbf{E}[\gamma_H(\mathbf{x}, \mathbf{y})] \right| > \epsilon_0 \right\}\right) \leq 2|\mathbb{H}| \exp(-2m\epsilon_0^2). \tag{A5}$$

By setting the right-hand side to a target value  $\eta$  and rearranging, we can find a sufficient number of samples is

$$m = \frac{1}{2\epsilon_0^2} \log \frac{2|\mathbb{H}|}{\eta}. \tag{A6}$$

In addition, solving  $m = \frac{1}{2\epsilon_0^2} \log \frac{2|\mathbb{H}|}{\eta}$  for  $\epsilon_0$  gives

$$\epsilon_0 = \sqrt{\frac{1}{2m} \log \frac{2|\mathbb{H}|}{\eta}}. \tag{A7}$$

We conclude that  $\left| \hat{\mathbf{E}}[\gamma_H(\mathbf{x}, \mathbf{y})] - \mathbf{E}[\gamma_H(\mathbf{x}, \mathbf{y})] \right| \leq \epsilon_0$ , for all  $H \in \mathbb{H}$ , with probability at least  $1 - \eta$ .

Now, assume that the preceding event occurs, and recall that  $H^*$  is the hyperparameter choice that minimizes  $\mathbf{E}[\gamma_H(\mathbf{x}, \mathbf{y})]$ . Then, we can further bound (A3) as follows:

$$\begin{aligned} \Delta_m &\leq \left| \hat{\mathbf{E}}[\gamma_{H^*}(\mathbf{x}, \mathbf{y})] - \mathbf{E}[\gamma_{H^*}(\mathbf{x}, \mathbf{y})] \right| + \left| \hat{\mathbf{E}}[\gamma_{\hat{H}}(\mathbf{x}, \mathbf{y})] - \mathbf{E}[\gamma_{\hat{H}}(\mathbf{x}, \mathbf{y})] \right| \\ &\leq \epsilon_0 + \epsilon_0 \\ &= 2\epsilon_0 \\ &= \sqrt{\frac{2}{m} \log \frac{2|\mathbb{H}|}{\eta}}. \end{aligned} \tag{A8}$$

This completes the proof of Theorem 1.

## References

1. Candès, E.J. Compressive Sampling. In Proceedings of the International Congress of Mathematicians, Madrid, Spain, 22–30 August 2006; Volume 3, pp. 1433–1452.
2. Bora, A.; Jalal, A.; Price, E.; Dimakis, A.G. Compressed Sensing Using Generative Models. In Proceedings of the 34th International Conference on Machine Learning, Sydney, Australia, 6–11 August 2017; pp. 537–546.
3. Ulyanov, D.; Vedaldi, A.; Lempitsky, V. Deep image prior. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Salt Lake City, UT, USA, 18–23 June 2018; pp. 9446–9454.
4. Heckel, R.; Hand, P. Deep Decoder: Concise Image Representations from Untrained Non-convolutional Networks. In Proceedings of the International Conference on Learning Representations, New Orleans, LA, USA, 6–9 May 2019.
5. Quan, Y.; Ji, H.; Shen, Z. Data-driven multi-scale non-local wavelet frame construction and image recovery. *J. Sci. Comput.* **2015**, *63*, 307–329. [\[CrossRef\]](#)
6. Van Veen, D.; Jalal, A.; Soltanolkotabi, M.; Price, E.; Vishwanath, S.; Dimakis, A.G. Compressed sensing with deep image prior and learned regularization. *arXiv* **2018**, arXiv:1806.06438.
7. Darestani, M.Z.; Heckel, R. Accelerated MRI with un-trained neural networks. *IEEE Trans. Comput. Imaging* **2021**, *7*, 724–733. [\[CrossRef\]](#)
8. Baguer, D.O.; Leuschner, J.; Schmidt, M. Computed tomography reconstruction using deep image prior and learned reconstruction methods. *Inverse Probl.* **2020**, *36*, 094004. [\[CrossRef\]](#)
9. Baldassarre, L.; Li, Y.H.; Scarlett, J.; Gözcü, B.; Bogunovic, I.; Cevher, V. Learning-Based Compressive Subsampling. *IEEE J. Sel. Top. Signal Process.* **2016**, *10*, 809–822. [\[CrossRef\]](#)
10. Budhaditya, S.; Pham, D.S.; Lazarescu, M.; Venkatesh, S. Effective anomaly detection in sensor networks data streams. In Proceedings of the IEEE International Conference on Data Mining, Miami, FL, USA, 6–9 December 2009; pp. 722–727.
11. Ravula, S.; Dimakis, A.G. One-dimensional Deep Image Prior for Time Series Inverse Problems. *arXiv* **2019**, arXiv:1904.08594.
12. Heckel, R. Regularizing Linear Inverse Problems with Convolutional Neural Networks. *arXiv* **2019**, arXiv:1907.03100.
13. Dhar, M.; Grover, A.; Ermon, S. Modeling sparse deviations for compressed sensing using generative models. In Proceedings of the International Conference on Machine Learning, Stockholm, Sweden, 10–15 July 2018.
14. Jagatap, G.; Hegde, C. Algorithmic guarantees for inverse imaging with untrained network priors. In Proceedings of the 33th Conference on Neural Information Processing Systems, Vancouver, BC, Canada, 8–14 December 2019; pp. 14832–14842.
15. Asim, M.; Daniels, M.; Leong, O.; Ahmed, A.; Hand, P. Invertible generative models for inverse problems: mitigating representation error and dataset bias. In Proceedings of the 37th International Conference on Machine Learning, Virtual Conference, 12–18 July 2020.
16. Yin, W.; Yang, W.; Liu, H. A neural network scheme for recovering scattering obstacles with limited phaseless far-field data. *J. Comput. Phys.* **2020**, *417*, 109594. [\[CrossRef\]](#)
17. Ongie, G.; Jalal, A.; Metzler, C.A.; Baraniuk, R.G.; Dimakis, A.G.; Willett, R. Deep learning techniques for inverse problems in imaging. *IEEE J. Sel. Areas Inf. Theory* **2020**, *1*, 39–56. [\[CrossRef\]](#)
18. Kattamis, A.; Adel, T.; Weller, A. Exploring properties of the deep image prior. In Proceedings of the NeurIPS 2019 workshop Deep Learning and Inverse Problems, Vancouver, QC, Canada, 13 December 2019.
19. Dittmer, S.; Kluth, T.; Maass, P.; Baguer, D.O. Regularization by architecture: A deep prior approach for inverse problems. *J. Math. Imaging Vis.* **2020**, *62*, 456–470. [\[CrossRef\]](#)
20. Uezato, T.; Hong, D.; Yokoya, N.; He, W. Guided deep decoder: Unsupervised image pair fusion. In Proceedings of the European Conference on Computer Vision, Glasgow, UK, 23–28 August 2020; Springer: Berlin/Heidelberg, Germany, 2020; pp. 87–102.
21. Heckel, R.; Soltanolkotabi, M. Compressive sensing with un-trained neural networks: Gradient descent finds a smooth approximation. In Proceedings of the 37th International Conference on Machine Learning, Virtual Conference, 12–18 July 2020.
22. Arora, S.; Roeloffs, V.; Lustig, M. Untrained modified deep decoder for joint denoising parallel imaging reconstruction. In Proceedings of the International Society for Magnetic Resonance in Medicine Annual Meeting, Virtual Conference, 8–14 August 2020.



23. Bostan, E.; Heckel, R.; Chen, M.; Kellman, M.; Waller, L. Deep phase decoder: Self-calibrating phase microscopy with an untrained deep neural network. *Optica* **2020**, *7*, 559–562. [[CrossRef](#)]
24. Rey, S.; Marques, A.G.; Segarra, S. An underparametrized deep decoder architecture for graph signals. In Proceedings of the 8th International Workshop on Computational Advances in Multi-Sensor Adaptive Processing (CAMSAP), Le Gosier, Guadeloupe, 15–18 December 2019; pp. 231–235.
25. Daniels, M.; Hand, P.; Heckel, R. Reducing the Representation Error of GAN Image Priors Using the Deep Decoder. *arXiv* **2020**, arXiv:2001.08747.
26. Darestani, M.Z.; Chaudhari, A.S.; Heckel, R. Measuring Robustness in Deep Learning Based Compressive Sensing. In Proceedings of the 38th International Conference on Machine Learning, Virtual Conference, 18–24 July 2021.
27. Zoph, B.; Le, Q.V. Neural architecture search with reinforcement learning. In Proceedings of the 5th International Conference on Learning Representations, Toulon, France, 24–26 April 2017.
28. Pham, H.; Guan, M.; Zoph, B.; Le, Q.; Dean, J. Efficient neural architecture search via parameters sharing. In Proceedings of the 35th International Conference on Machine Learning, Stockholm, Sweden, 10–15 July 2018.
29. Liu, H.; Simonyan, K.; Yang, Y. DARTS: Differentiable architecture search. In Proceedings of the 7th International Conference on Learning Representations, New Orleans, LA, USA, 6–9 May 2019.
30. Elsken, T.; Metzen, J.H.; Hutter, F. Neural architecture search: A survey. *J. Mach. Learn. Res.* **2019**, *20*, 1997–2017.
31. Jamieson, K.; Talwalkar, A. Non-stochastic best arm identification and hyperparameter optimization. In Proceedings of the 18th Artificial Intelligence and Statistics, Cadiz, Spain, 9–11 May 2016; pp. 240–248.
32. Li, L.; Jamieson, K.; DeSalvo, G.; Rostamizadeh, A.; Talwalkar, A. Hyperband: A novel bandit-based approach to hyperparameter optimization. *J. Mach. Learn. Res.* **2017**, *18*, 6765–6816.
33. Ioffe, S.; Szegedy, C. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In Proceedings of the 32nd International Conference on Machine Learning, Lille, France, 6–11 July 2015; pp. 448–456.
34. Shalev-Shwartz, S.; Ben-David, S. *Understanding Machine Learning: From Theory to Algorithms*; Cambridge University Press: Cambridge, UK, 2014.
35. Dirksen, S.; Jung, H.C.; Rauhut, H. One-bit compressed sensing with partial Gaussian circulant matrices. *Inf. Inference A J. IMA* **2020**, *9*, 601–626. [[CrossRef](#)]
36. Rasmussen, C.E. *Gaussian Processes for Machine Learning*; MIT Press: Cambridge, MA, USA, 2006.
37. De Vito, S.; Massera, E.; Piga, M.; Martinotto, L.; Di Francia, G. On field calibration of an electronic nose for benzene estimation in an urban pollution monitoring scenario. *Sens. Actuators B Chem.* **2008**, *129*, 750–757. [[CrossRef](#)]
38. Liu, Z.; Luo, P.; Wang, X.; Tang, X. Deep Learning Face Attributes in the Wild. In Proceedings of the International Conference on Computer Vision (ICCV 2015), Santiago, Chile, 11–18 December 2015.
39. Lustig, M.; Donoho, D.L.; Santos, J.M.; Pauly, J.M. Compressed sensing MRI. *IEEE Signal Process. Mag.* **2008**, *25*, 72–82. [[CrossRef](#)]
40. Zbontar, J.; Knoll, F.; Sriram, A.; Murrell, T.; Huang, Z.; Muckley, M.J.; Defazio, A.; Stern, R.; Johnson, P.; Bruno, M.; et al. fastMRI: An open dataset and benchmarks for accelerated MRI. *arXiv* **2018**, arXiv:1811.08839.
41. Sheikh, H.R.; Bovik, A.C. Image information and visual quality. *IEEE Trans. Image Process.* **2006**, *15*, 430–444. [[CrossRef](#)] [[PubMed](#)]
42. Wang, Z.; Bovik, A.C.; Sheikh, H.R.; Simoncelli, E.P. Image quality assessment: From error visibility to structural similarity. *IEEE Trans. Image Process.* **2004**, *13*, 600–612. [[CrossRef](#)] [[PubMed](#)]
43. Wang, Z.; Simoncelli, E.P.; Bovik, A.C. Multiscale structural similarity for image quality assessment. In Proceedings of the 37th Asilomar Conference on Signals, Systems & Computers, Pacific Grove, CA, USA, 9–12 November 2003; Volume 2, pp. 1398–1402.
44. Boucheron, S.; Lugosi, G.; Massart, P. *Concentration Inequalities: A Nonasymptotic Theory of Independence*; Oxford University Press: Oxford, UK, 2013.