ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

CVL Computer Vision Lab

Exploring Deep Decoder applied to MRI

Semester Project

Pablo Blasco Fernández
Department of Information Technology and Electrical Engineering

Advisors: Jakob Geusen and Gustav Bredell
Supervisor: Prof. Dr. Ender Konukoglu

July 25, 2023

Abstract

Untrained neural networks have emerged as a potential solution to address the scarcity of large datasets in deep learning, as they do not rely on previous learning. This characteristic can prove valuable in domains where a trade-off exists between training and test data or when labeled data is not readily available. In this project, we explore the capabilities of the untrained model called the Deep Decoder.

Our analysis involves introducing variations in the model's settings to gain insights into their impact on performance, specifically focusing on inference time, stability and the quality of its outputs so that one could asses how useful this model can be for further specific tasks.

After using variations of the inputs and targets of the model, and specifically by using MRI slices, we analyze how different hyperparameters play a role. Our research found that indeed choosing the set of hyperparameters conditions the robustness of the model. Understanding these relationships can aid in optimizing the model's performance and guiding its application.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In this chapter we first introduce the motivation of the project in Section 1.1, discussing deep learning trained neural networks, where large labeled datasets are traditionally required. However, challenges arise in domains like medical imaging due to limited data availability. Recent research explores untrained networks as "free-training" models, utilizing their architecture as image priors and minimizing loss functions based solely on observed data. This approach shows promise in overcoming data scarcity and avoiding domain adaptation needs. In Section 1.2, we introduce these untrained models, followed by a presentation of specific networks related to the project in Section 1.3. and the focus of all these in Section 1.4 to end up describing how this report is organized in Section 1.5.

1.1 Motivation

Deep learning neural networks (NNs), both in supervised and unsupervised settings, are characterized by their reliance on a large number of training samples from labeled datasets [BOS]. These networks leverage extensive data to uncover intricate patterns and complexities, enabling accurate performance in various tasks. However, in domains such as medical imaging, generating large-scale datasets faces challenges in terms of feasibility, time-consuming manual annotations by experts, and concerns regarding patient privacy and safety (e.g., ionizing radiation imaging) [DSHG$^+$21].

One might initially assume that the performance of these models is mostly reliant on the quality and quantity of training data. However, Zhang et al [ZBH$^+$21] challenge this conventional understanding by demonstrating that the network's size and architecture significantly contribute to its generalization performance.

Given these insights, recent research has focused on the development of untrained networks [GSI, HH19, QIS$^+$23, UVL20, WBW$^+$20], which operate without the need for traditional training data. Instead, these "free-training" models rely on their inherent architecture as image priors and are fitted by minimizing a loss function defined purely based on observed data.

This might be a promising research line in order to tackle the scarce of huge datasets for training and furthermore, avoid the need of domain adaptation given that the network accommodates to the input data, specifically in inverse imaging problems.

1.2 Untrained Neural Networks

When training neural networks in inverse problems such as restoration [LGG+], denoising [VLL+10], inpainting [QZZX21] or super-resolution [DLHT16], the objective[1] as defined in [JH] is to find a mapping $x \to y$ represented by $y_n = f(x; \theta^*)$. Here, y refers to the ground truth image, x represents the input to the network (which is usually a corrupted image), and $f : x \to y \in \mathbb{R}^n$ denotes the function that maps x to y with the optimal hyperparameters θ^*.

In order to obtain the desired restoration, it becomes necessary to determine the closest set of hyperparameters θ that characterize f, and thus approximate to θ^*. This typically involves solving an optimization problem, which can be formulated as follows:

$$\theta^* = arg_\theta min \sum_{n=1}^{N} \|y_n - f(x_n; \theta)\|_2^2 ;\, y_n \approx f(x_n; \theta^*) \tag{1.1}$$

After a number of iterations in the training process, the parameters of the network θ are optimized to minimize the loss between the ground truth and the corrupted image and hence resemble as close as possible to $\theta*$.

However, in the case of untrained networks, and specifically in the Deep Image Prior [UVL20] and Deep Decoder [HH19] (explained in detail in 1.3) in which this work is focused, we have a different optimization problem:

$$\theta^* = arg_\theta min \sum_{n=1}^{N} \|x_n - f(z_n; \theta)\|_2^2 ;\, y_n \approx f(z_n; \theta^*) \tag{1.2}$$

Here, z is a random fixed input and the goal is to fit the data to the network so that $x_n \approx f(z_n; \theta^*)$ with the hope that the implicit image prior of the untrained network will lead to $y_n = f(z_n; \theta^*)$ since the network is underparametrized so that the noise would not be fitted and hence the ground truth image can be recovered as shown in [HH19].

1.3 Deep Image Prior and Deep Decoder

Ulyanov et al. [UVL20] introduced the Deep Image Prior (DIP), an encoder-decoder architecture based on deep convolutional neural networks (CNNs). DIP was the pioneering 'free-training' model, exhibiting good reconstruction performance without training. This overparametrized network's inherent design and regularization through early stopping enabled it to effectively capture intricate image priors.

Furthermore, Heckel et al. [HH19] observed that simplifying the model by removing skip connections and retaining only the decoder architecture achieved notable results. They named this modified architecture the Deep Decoder (DD). As an underparametrized network, DD maps a lower-dimensional space to a higher-dimensional space, resembling classical image representations such as sparse wavelets [SMF]. The advantage of this is its inherent barrier against overfitting, enabling effective regularization for inverse problems (More on this in Section 2.2 Deep Decoder Insights). DD serves as a natural data model and, similar to the Deep Image Prior (DIP), it does not require training but also nor relies on regularization and early stopping. This property offers

[1] In this section we describe the optimization problem using the notation presented in [JH], as they analyze the theoretical guarantees of this type of models. Nonetheless, Heckel [HH19] defines it differently. We adopt this section's notation as a general view of the problem and use specific notation for the Deep Decoder in subsequent sections.

the benefit of versatility in application and insensitivity to potential discrepancies between training and test data.

Unlike traditional networks for image compression, restoration, and recovery, that employ convolutional layers with nontrivial spatial filters, the DD does not use convolutions. Instead, it employs pixelwise linear combinations of channels, with shared weights among spatial positions. Although sometimes referred to as '1x1 convolutions', these operations lack spatial coupling between pixels. DD demonstrates that relationships characteristic of neighboring pixels in natural images can be imposed directly through upsampling layers, showcasing an alternative approach to incorporating spatial dependencies.

While research on the Deep Image Prior (DIP) has explored its application across various tasks and domains, often in conjunction with other techniques [GSI, LSXK, VVJS$^+$, MEM, JH], it is noteworthy that only one publication has focused on applying the DIP to medical images [YJG$^+$]. Furthermore, limited extensive work has been conducted on the application of the Deep Decoder to medical images or other types of natural images nor an intuition has been developed about its behavior when more than one image has to be fit or the hyperparameters of the model vary.

1.4 Focus of this Work

The objective of this project is to investigate the capabilities of the DD with medical imaging, specifically for representations of magnetic resonance (MR) images, while also developing insights into the effect of hyperparameters to achieve efficient results, which could further be used on specific inverse problems tasks such as denoising, super-resolution or even for segmentation. Although DD has demonstrated promise as a proof of concept, its practical application remains limited. Unlike trained networks, DD requires significant inference time as it needs to fit to each individual image. Consequently, further research is needed to enhance its computational efficiency and expand its widespread usability for images with certain complexity in intensity and geometry such as MR.

In the present work, DD parameters are tunned in order to understand how the behaviour of the DD changes in terms of inference time, stability and output quality. Moreover we experiment with different ways of fitting and adapt so that it can fit for several images at once. Moreover, we experiment with the nature of the random input tensor z that is fed to the network to get the output of the model.

1.5 Thesis Organization

After this introduction, Chapter 2 delves into the related work. It is important to note that comparing this untrained model with the current state-of-the-art work may prove challenging. Nevertheless, we contextualize the insights of the Deep Decoder architecture in the current state of the art.

Chapter 3 presents the methodology adopted to optimize the model's performance and develop an intuition on how certain modifications can improve its results in representing the target images. This involves exploring numerous combinations of hyperparameters for the network and experimenting with input random tensors of various characteristics. Additionally, we examine different ways of presenting the target image to the Deep Decoder to test weather it could fit several images at a time.

In Chapter 4, the experimental results are presented both qualitatively and quantitatively, providing a comprehensive analysis of the outcomes.

Chapter 5 concludes with a detailed discussion of the results and offers insights into potential future research.

Chapter 2

Related Work

This chapter provides an overview of the existing literature on inverse problems in imaging. Section 2.1 discusses the traditional methods for solving inverse problems, dividing them in categories that are explained in its subsections. Section 2.2 presents the insights of the Deep Decoder architecture, which is the model used in this work.

2.1 Deep Learning for Inverse Problems

Inverse imaging problems (IIPs) such as denoising or inpainting have become a prominent area of research, finding applications in various fields with medical imaging among them. In these, the observations are obtained through a forward process that is ill-posed, resulting in multiple possible solutions for the same problem. However, finding a unique solution that accurately matches these observations is difficult or even impossible without having reliable prior knowledge about the data. Traditional algorithms for IIPs rely on manually crafted mathematical models based on domain knowledge [EHN]. However, this approach has a disadvantage as it requires specificity to a particular domain to achieve good performance. In contrast, deep learning-based techniques for inverse problems have demonstrated greater effectiveness and superior performance [OJM$^+$]. As said, untrained networks can be seen as a compromise between the need of large datasets of traditional models and the need of specific algorithms of handcrafted. To give an overview of the state of the art (See Figure 2.1) we show the different strategies for dealing with inverse problems as described in [QIS$^+$23] in the following subsections.

2.1.1 Methods with training

These methods heavily rely on labeled data (ground truth) to achieve good performance. However, they do come with certain drawbacks [LTMK], most notably the need for a substantial amount of training examples. Acquiring such data can be challenging and costly, especially in domains like medical imaging. Additionally, these methods may have limited generalization ability when applied to different types of data.

Despite these limitations, trained methods remain prevalent in the state of the art due to their fast inference time once training is completed and their capability to deliver optimal results. The ability to automatically learn features from the data and achieve state-of-the-art performance on specific tasks makes trained methods a valuable and widely used approach in various image reconstruction and analysis tasks [ZYB]. The two most common approaches are dictionary learning along with deep learning based models:

- On the one hand, in the dictionary learning approach [ZXY+, ZYB], we utilize the training data to learn a set of representative images called a dictionary. This dictionary is designed to capture essential patterns and features present in the training data. The idea is that by using this learned dictionary, we can efficiently represent a given image as a combination of a few elements from the dictionary. Given an input image, we aim to reconstruct it using only a small number of dictionary elements, which form a sparse representation of the image. By carefully selecting and learning the dictionary from a diverse set of training images, we can represent a wide range of image structures effectively.

- On the other hand, deep learning-based methods for inverse imaging problems leverage the power of trained deep NNs to address task-specific challenges [OJM+]. They usually approach the problem as explained in Section 1.2, equation 1.1, that is, obtaining the mapping $f : x \to y \in \mathbb{R}^n$, which allows the recovery of the ground truth image by training with lots of examples. The goal is to minimize the loss between the guessed output and the labelled example by optimizing the weights of the model. The most common examples of this are generative models [ZYB].

Since these approaches are task-specific, the network becomes good at that particular task, but it may not work well for other goals. For example, if we train a deep learning network to denoise images, it will become specialized in denoising, but it may not perform as well if we want it to do super-resolution, since it learns to recognize certain patterns and features in the data related to denoising. Hence, the retraining process can be time-consuming and computationally expensive.

2.1.2 Methods without training

Dealing with inverse problems in imaging without training involves employing "untrained" or "prior-based" methods that do not depend on pre-existing labeled datasets. These approaches aim to reconstruct or enhance images directly, without the need for task-specific training. The two main categories, as described in [QIS+23], include handcrafted priors and untrained neural networks, with the latter serving as the basis for this project:

- In the case of handcrafted priors for inverse problems, they utilize prior knowledge about the true image, such as sparsity, smoothness, total variation, non-local self-similarity, or other geometric properties [HZL+]. The goal is to find a solution that fits the observed data well, given this prior knowledge. The most common example for these, also used as a reference to compare with untrained NNs in [HH19] is the wavelet decomposition [Mou] by which when we say that the image is "modeled in the linear span of a few elements in a space" [QIS+23], meaning that we can represent the image using a combination of wavelet coefficients. However, it is essential to understand that this representation may not always be the most accurate or best way to describe the image since it relies on linear combinations and hence it might not capture all the fine details and complexities of the original image.

- To finalize this categorization, we refer to untrained neural networks. Rather than training deep learning models on large-scale datasets, untrained deep learning models are employed to model natural images. In this approach, the DL model is initialized with random parameters and provided with random noise as input. The task is then to use this random input and its parameters to model a specific natural image. As explained in Section 1.3, the first contribution to this field which gave comparable results with state of the art NNs was the Deep Image Prior (DIP) by Ulaynov [UVL20], with some applications already proven. However, it does have limitations, as it relies heavily on strong regularization and early stopping. This

can make it challenging to use the model for automatic tasks and might not be the most time-efficient option. Moreover Qayyum [QIS+] has highlighted the need of optimizing untrained NNs to achieve optimal results in a more time-efficient manner. The Deep Decoder (DD) unique architecture makes it simpler and could potentially eliminate the need for extensive regularization, making it a promising alternative for inverse imaging problems. By exploring the capabilities of the Deep Decoder, researchers can gain insights into its effectiveness in various imaging tasks, such as denoising, super-resolution, inpainting, etc. Understanding its performance in different scenarios can provide valuable information for optimizing and fine-tuning the model for specific applications, and still seems an unexplored research topic.

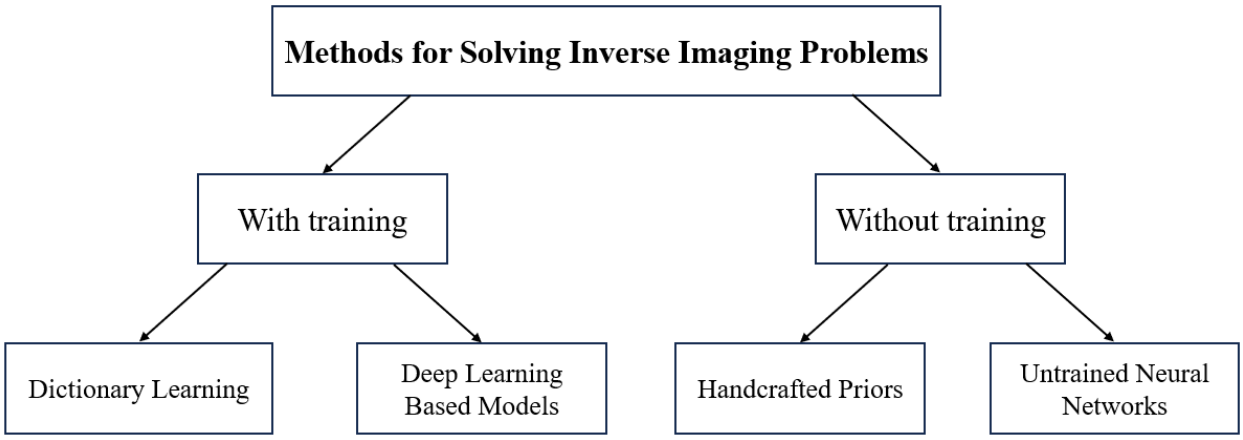


Figure 2.1: The schematic illustrates the classification of strategies employed to address inverse problems in imaging. These approaches are categorized based on whether they require training or not.

2.2 Deep Decoder Insights

As discussed in Section 1.3, we explore the Deep Decoder (DD)[1], a neural network that serves as an underparameterized, untrained, yet effective image prior for image processing. Unlike traditional approaches that rely on training NNs with image pairs, the proposed method involves directly fitting the model to a (corrupted) image $\mathbf{y}$ to obtain a ground truth image $\mathbf{x}$ with $\mathbf{y} = \mathbf{f}(\mathbf{x}) + \eta$, where $\mathbf{f}$ represents a forward operator and $\boldsymbol{\eta}$ denotes any type of noise.

The fitting process commences by randomly initializing the neural network's weights $\mathbf{C} = \{\mathbf{C}_0, \mathbf{C}_1, \ldots, \mathbf{C}_{d-1}\}$, where $\mathbf{C}i \in \mathbb{R}^{k_i \times k_{i+1}}$, and k_i represents the number of channels in the i-th layer of the network. However, in this work, we set the number of channels ($\boldsymbol{k}$) of every layer to the same fixed value. The input tensor $\mathbf{B}_0 \in \mathbb{R}^{w \times h \times k0}$ (fixed) undergoes multiple blocks $\boldsymbol{l}$, each consisting of a sequence of operations: convolutions with a 1x1 kernel conv$(\cdot)$, upsampling with bi-linear interpolation $\mathbf{U}_i$, rectified linear units ReLU$(\cdot)$, and batch normalization bn$(\cdot)$ until it reaches the size of the target image. The output of this process is an image $\hat{\boldsymbol{x}} \approx \boldsymbol{x}$ (See Figure 2.3):

$$B_{i+1} = \text{bn}\left(\text{ReLU}\left(\mathbf{U}i\left(\text{conv}(Bi, \mathbf{C}_i)\right)\right)\right), \quad i = 0, \ldots, d-1. \tag{2.1}$$

[1]Here, we use the notation described in [HH19]

Afterward, the tensor goes through two final operations to produce the output $\hat{\boldsymbol{x}}$: a convolution layer that reduces the number of channels from k to 1 for single-channel images or 3 for RGB images, and a sigmoid activation:

$$\hat{x} = \text{sigmoid}\left(\text{conv}(B_d, C_d)\right) \tag{2.2}$$

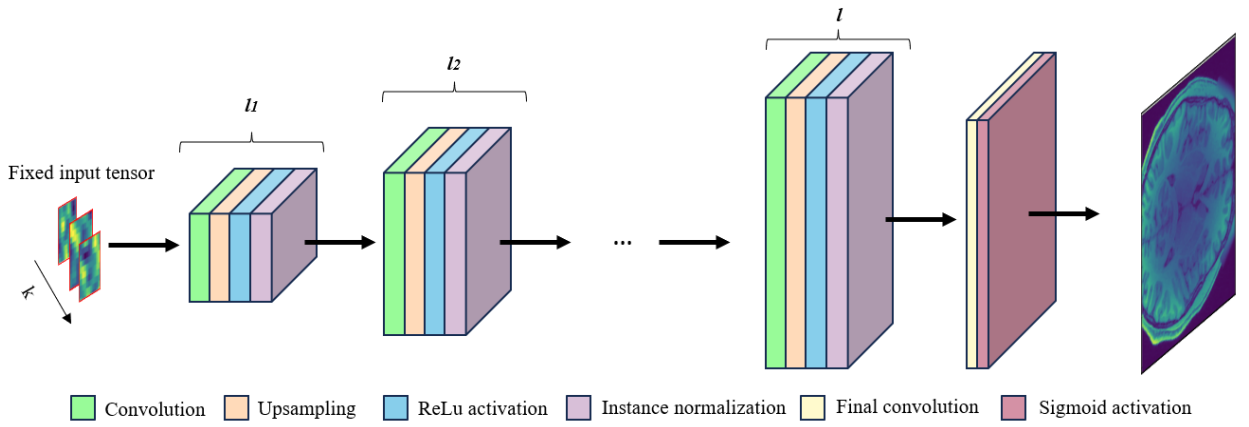


Figure 2.2: Deep Decoder architecture, composed by n blocks of a sequences of convolutions, umsampling, ReLU activations and normalization to finish with a 1x1 convolution and a sigmoid activation to output the final image

The simplicity of the model allows to investigate and develop an intuition on how tuning the hyerparameters of the network might lead to efficient results. These hyperparameters are the **number of iterations** for the fitting, the random input size that undergoes subsequent upsampling up to the output image size (determined by a scaling factor $\boldsymbol{s}$ with respect to the target size), the number of channels $\boldsymbol{k}$ and the number of blocks of consecutive layers $\boldsymbol{l}$.

Then, the weights are optimized by minimizing the loss between the input image and the output of the Deep Decoder, typically using the Adam optimizer. The network's mapping of the weights is denoted as $\boldsymbol{G(C)} = \hat{x} \approx x$ (See Figure 2.2):

$$L(C) = \|G(C) - x\|_2^2 \tag{2.3}$$

The number of weights of this underparametrized network (number of parameters to be optimized to minimize the loss) are calculated as follows (note that neither activation nor upsampling layers have optimizable parameters, so we do not need to account for them in the calculation):

1. For each block l_i where i is in the range $[1, d-1]$, with k fixed number of input and output channels:
   - Weights in the 1x1 kernel convolution layer (without bias): $k \times k \times 1 \times 1 = k^2$
   - Weights in the instance normalization layer (with scaling and bias): $k \times 2 = 2k$
   - Total number of parameters per block l_i: $k^2 + 2k$

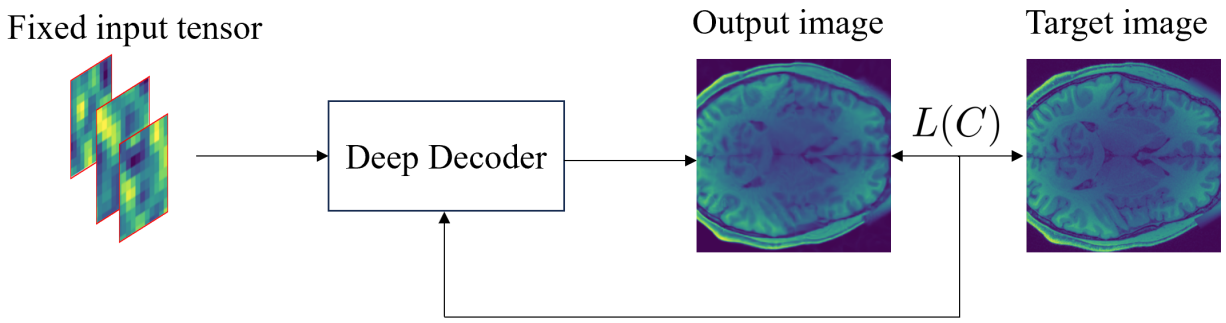


Figure 2.3: Fitting of the Deep Decoder. A fixed random input is given, for which a loss between the target image and the output is computed that to optimize the weights C_i of the network

2. For the final convolution layer from k input channels to 1 (greyscale image) and 1x1 kernel: $k \times 1 \times 1 \times 1 = k$

Combining these for all blocks and the final layer, we get:

$$\text{Total number of parameters} = l \times (k^2 + 2k) + k \tag{2.4}$$

With l being the total number of blocks. This formula holds for our implementation of the Deep-Decoder in this project. For instance, for $k = 64$ and $l = 5$ we get:

$$\text{Total parameters} = 5 \times (64^2 + 2 \times 64) + 64 = 21184$$

Given an image of size (256×320), the network maps 21184 parameters into 81920 pixel values, i.e. it compresses the information at a scale of $\frac{21184}{256 \times 320} = \frac{21184}{81920} = 0.257$. Following on this, the model might not fully capture the complexity of the target image, leading to some information loss based on the network's design. However, this loss of information primarily relates to the actual corruption in the image, typically noise. Consequently, the output often results in a reconstruction of the ground truth image when the parameters are well chosen.

It is important to note that this reconstruction of the ground truth image can then only capture high-frequency features up to a certain level. Since the model cannot account for noise, it is also incapable of accurately representing fine details of that scale present in the ground truth image.

Another important aspect to consider is the random initialization of the input tensor, introducing randomly distributed noise. While this may add variability to the process, it can also influence the final outcome of the reconstruction for each initialization.

Chapter 3

Method

This chapter provides a detailed description of the methodology used in the project. It begins with Section 3.1, which discusses the input tensors used for fitting the Deep Decoder. These input tensors comprise three different types of noise, which play a crucial role in the DD fitting process. Moving on to Section 3.2, we delve into the presentation of target images, which serve as the ground truth reference during the training of the Deep Decoder. Understanding how the target images are presented to the DD is useful to grasp the model's learning process. Section 3.3 presents the hyperparameter tuning made to the model. Finally, Section 3.4 extensively covers the metrics utilized to assess the quality of the output images generated by the Deep Decoder.

3.1 Input Tensors

In the fitting of the Deep Decoder (DD), both a fixed random input $\boldsymbol{B_0}$ and the network's weights $\boldsymbol{C}$ are randomly initialized. To investigate the impact of the structure of this random tensor on image generation, we experimented with three variations of noise. Our hypothesis was that the spatial correlation between the pixels in the input tensor might be beneficial for generating spatially correlated images, such as natural images, which, to the best of our knowledge, has not been explored in the literature yet. The three types of noise used as inputs are randomly and uniformly distributed noise, Gaussian-blurred noise, and a variation of Perlin noise [Bur] called Simplex noise [Per].

3.1.1 Random Uniform Noise and Gausian-blurred Noise

Since the DD takes any input image for which the network is fitted to a target image, the standard practice is to just take any random input without considering its structure. In our work we follow the same procedure and provide as input a random image of uniformly distributed noise, where all the values within a range have the same probability to appear (We will refer to this type of noise when talking about Blur = 0). Since we normalize the target image between [0, 1], we set the range of values for the noise to also be [0, 1]. However, we found that using a range of values between [-1, 1] did not yield significantly different results. Since all values in the uniformly distributed noise are equally likely, there is no inherent relationship between neighboring pixels. Hence, blurring this same noisy input with a Gaussian kernel might be able to introduce this relationships between pixels (See Figure 3.1).

The formula for a Gaussian blurring kernel used in the blurring process is given by:

$$G(x, y) = \frac{1}{2\pi\sigma^2} \cdot e^{-\frac{x^2+y^2}{2\sigma^2}} \tag{3.1}$$

Where $G(x, y)$ is the value of the Gaussian kernel at position (x, y) and σ is the standard deviation controlling the amount of blurring.

However, images are represented in a discrete grid of pixels,it is needed to convert the continuous Gaussian function into a discrete kernel. To achieve this, we sample the Gaussian function at specific intervals within a 3x3 neighborhood around the central pixel (x, y). When talking about 'Blur = 3' in other sections we will refer to this type of noise.

The central pixel corresponds to (0, 0) in the kernel, and the other positions within the 3x3 neighborhood are $(-1, -1)$, $(-1, 0)$, $(-1, 1)$, $(0, -1)$, $(0, 1)$, $(1, -1)$, $(1, 0)$, and $(1, 1)$. Each of these positions represents a square in the 3x3 grid.

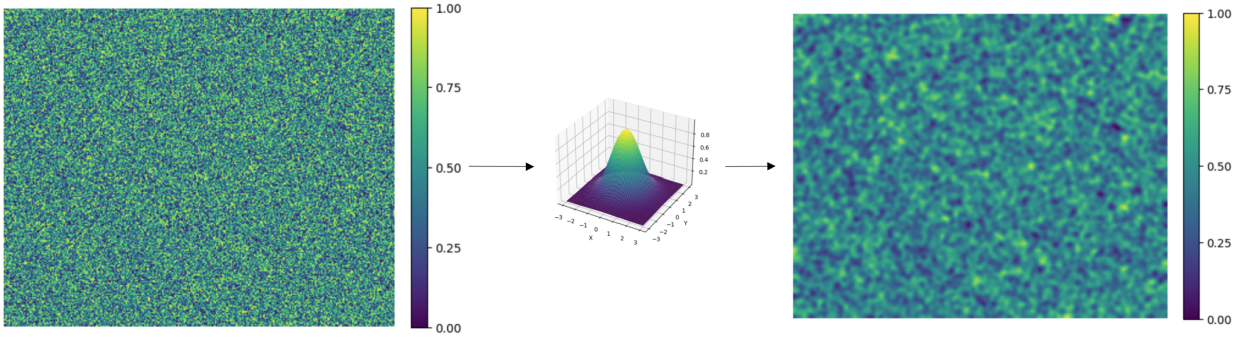
For each square in the grid, we calculate the value of the Gaussian function using the formula for $G(x, y)$ with the corresponding (x, y) position relative to the central pixel. These calculated values form the discrete Gaussian kernel.

The resulting values for the 3x3 Gaussian kernel are as follows:

$$\begin{bmatrix} G(-1,-1) & G(-1,0) & G(-1,1) \\ G(0,-1) & G(0,0) & G(0,1) \\ G(1,-1) & G(1,0) & G(1,1) \end{bmatrix}$$

Each value in this 3x3 matrix represents the weight or influence of the corresponding square in the grid on the blurring process. These weights determine how much neighboring pixel information is considered when blurring the central pixel.

By adjusting the standard deviation (σ) of the Gaussian function, we control the spread of the Gaussian curve and, in turn, the extent of blurring applied to the image. A larger σ results in a wider curve and more smoothing, while a smaller σ results in less blurring. The standard deviation directly influences the shape of the Gaussian function and, consequently, the values in the discrete Gaussian kernel.



(a) Random Uniformly Distributed Noise

(b) Gaussian-blurred Noise

Figure 3.1: Input noise to the DD. On the left (a) we can see a example of random uniformly distribute noise [0,1]. On the right side in (b), we can observe the result of applying a Gaussian kernel of size 3x3 to the same input noise.

After some tests, our experiments showed best results when the standard deviation σ was calculated automatically from the kernel size of the filter by the following formula provided by the OpenCV library in Python:

$$\sigma = 0.3\,((\text{kernel size} - 1) \cdot 0.5 - 1) + 0.8 \tag{3.2}$$

However, in all experiments involving the blurred input, a kernel size of 3x3 was used ($\sigma = 0.8$ according to Equation 3.1). The decision to use this kernel size was due to the initial tensor's small

size before upsampling, which made it impractical to use larger kernels. Using larger kernels in such a scale would result in excessively aggressive blurring, rendering the images unrecognizable.

3.1.2 Perlin and Simplex Noise

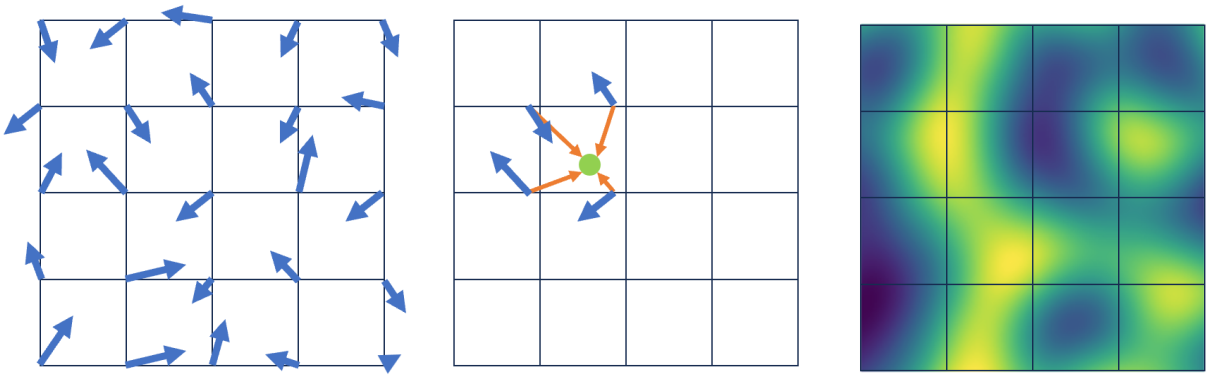
Perlin noise, developed by Ken Perlin in 1983 [Per85], is a gradient-based noise function used in procedural texture generation. This algorithm combines multiple layers of noise with varying frequencies and amplitudes to produce natural-looking textures with a random, organic appearance. Its ability to create smooth, continuous noise by interpolating between randomly generated gradient vectors makes it a valuable tool in computer graphics, animation, and game development. Over the years, Perlin noise has gained widespread popularity and is commonly employed to generate realistic textures for various applications, including simulation of breast tissue texture [DBP+]. The way Perlin noise values are calculated for a grid of a given size is as follows (See Figure 3.2 to illustrate the process):

(1) Grid Definition: Perlin noise starts with defining a grid of random gradient vectors. Each grid intersection has an associated fixed random n-dimensional gradient vector. For example, in two dimensions, the grid forms a square with each corner having a random 2D vector.

(2) Dot Product: To compute the value of a point within the grid, a unique grid cell in which the point lies is found. Then, an offset vector is calculated from each corner of the cell to the candidate point. For each corner, the dot product between its gradient vector and the offset vector is taken. The dot product will be zero if the candidate point is exactly at the grid corner. This step effectively measures the influence of each corner's gradient vector on the candidate point.

(3) Interpolation: The final step is interpolation between the dot products of the gradient vectors. Interpolation is performed using a function that has zero first derivative at the grid nodes. This ensures that the output will approximate the dot product of the gradient vector of the node and the offset vector to the node, giving Perlin noise its characteristic smooth and continuous look. A popular interpolation function used is the smooth step function in 1D or bilinear interpolation in 2D.

However, the original implementation of this noise presented some drawbacks, namely its unnaturally looking appearance and a large generation time. To overcome these, a variation of the original Perlin noise was proposed called perlin Simplex noise [Per]. While Perlin noise is based on a regular grid of gradient vectors and creates smooth patterns by interpolating between these vectors, Perlin Simplex noise uses a simplex (triangular) grid, which reduces computational complexity and improves performance. It also produces smoother and more natural-looking patterns compared to Perlin noise. Although we illustrate the process with Perlin noise for clarity, we use Simplex Perlin noise implementation as the input. Interestingly, the available implementation of Simplex noise generates a random sample of n-dimensional noise and then calculates the value of the pixel in each of the coordinates of the grid. The scale or frequency of the generated noise is controlled by the feature size, which determines how closely spaced the random gradients are. A larger feature size produces larger-scale patterns, while a smaller one leads to finer details and smaller-scale structures in the resulting noise images (See Figure 3.3). In our experiments we use feature size of 1 since the input size are small, and other values for the feature size should be chosen accordingly as in the case of the sigma value σ for the Gaussian-blurred noise.

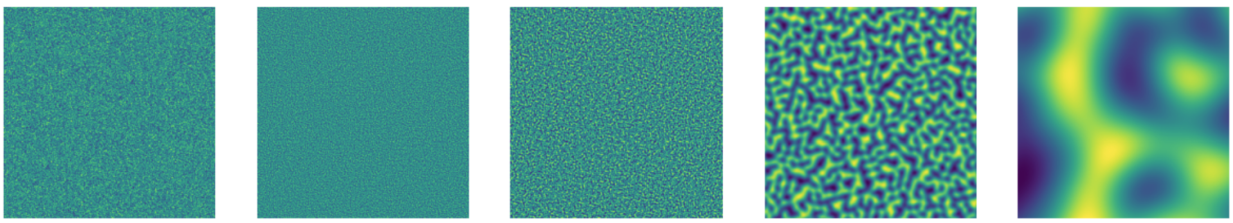


(1) Define a grid, divide it in cells and obtain a random gradient vector

(2) Dot product between gradients and offset vectors

(3) Interpolate values of the dot products for every point

Figure 3.2: Generating Perlin noise involves three key steps: (1) Grid Definition: Random gradient vectors are assigned to grid intersections forming a square in 2D. (2) Dot Product: Select each pooint, calculate the offset vector from the corners and do the dot product between offset and gradient vectors (3) Interpolation: Interpolate dot products of gradient vectors using a function with zero first derivative at grid nodes, achieving Perlin noise's smooth and continuous appearance.



(a) Feature size = 0.5 (b) Feature size = 1 (c) Feature size = 2 (d) Feature size = 10 (e) Feature size = 100

Figure 3.3: The larger the feature size, the larger the scale of the noise, resulting in larger patterns or structures in the generated images. Conversely, a smaller feature size will lead to finer details and smaller-scale patterns.

This type of noise seems to be a compromise between the totally uncorrelated random uniform noise and the Gaussian-blurred noise. We will refer to Simplex Perlin noise just by talking about 'Perlin' from now on.

3.2 Target Images

The main objective of this project was to assess the performance and behavior of the DD model on complex medical images, specifically magnetic resonance (MR) images. To achieve this, we selected a T1 brain mid-slice as the target image and used it to fit the DD (using as input the three types of noise of the previous section). We conducted experiments on several slices from different subjects, and the results were consistent across the slices. To enhance clarity, we present the results for a single slice in this report.

Additionally, we were interested in understanding the DD capabilities to handle multiple images simultaneously. To explore this, we also fitted the model on the slice divided into grids of patches

with sizes 4x5, 3x3, and 2x2. This approach allowed us to observe how the performance varied when capturing and representing information at a smaller scale (See Figure 3.4).

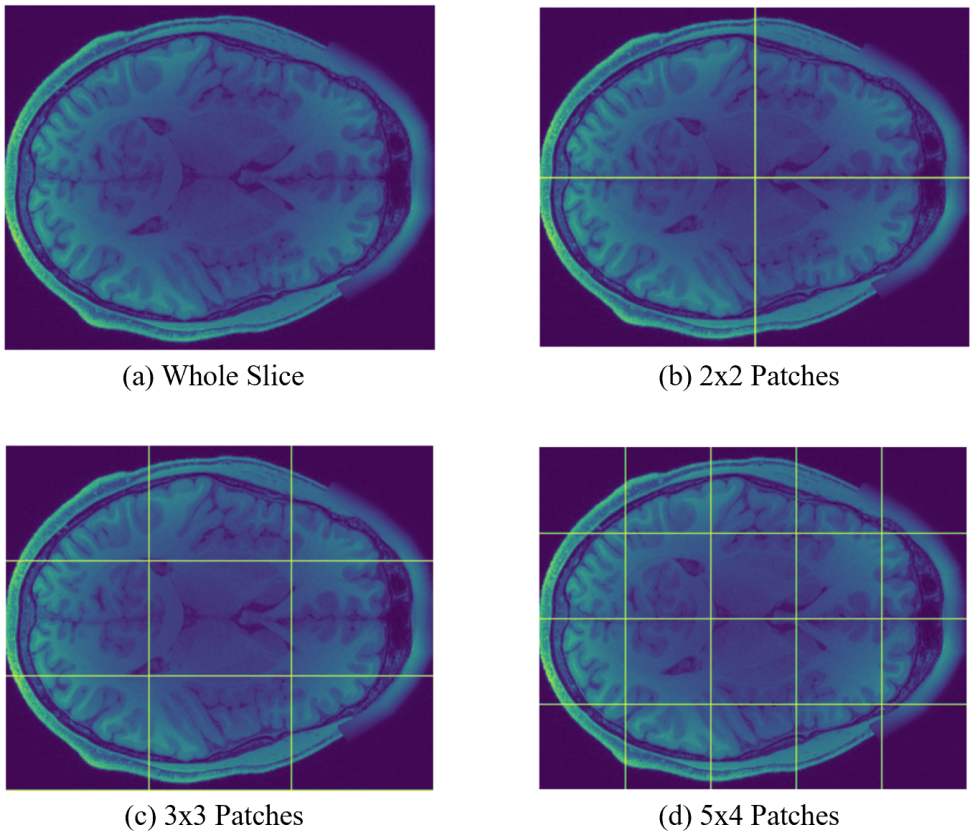


Figure 3.4: The figure shows a brain T1 magnetic resonance (MR) mid slice image (a) followed by its division into grids of (b) 2x2 patches, (c) 3x3 patches (bottom-left), and (d) 4x5 patches. The patch division enables a deeper understanding of the neural network's capabilities in handling several multi-scale inputs. Note that all images are grayscale (1-channel) and a colormap was applied to the pixel intensities to enhance the discrimination of structures.

To accommodate multiple images for fitting, we made a modification to the original DD model by enabling it to accept several images at once with a variable batch size, necessitating the use of instance normalization instead of batch normalization layers. The instance normalization was employed to normalize channel-wise for all the images, as the original architecture was designed for a batch size of 1.

During the fitting process, instead of feeding the neural network with randomly generated noise inputs per image (the whole image or each patch), we created a sample of noise of a specific input shape — in this project, we generated noise of the size of the target image and then scaled it down by dividing its size by a scaling factor $\boldsymbol{s}$ so that it undergoes upsampling up to the target image size and then divided it into patches in the same manner as the image. This approach ensured the preservation of spatial correlation between the pixels of the noise inputs, maintaining the contextual information during the fitting process (See Figure 3.5).

Moreover, we devised a way to create "3D" noise inputs for our model, where each noise sample corresponds to a hidden channel in the DD. The process involved using two noise samples, namely

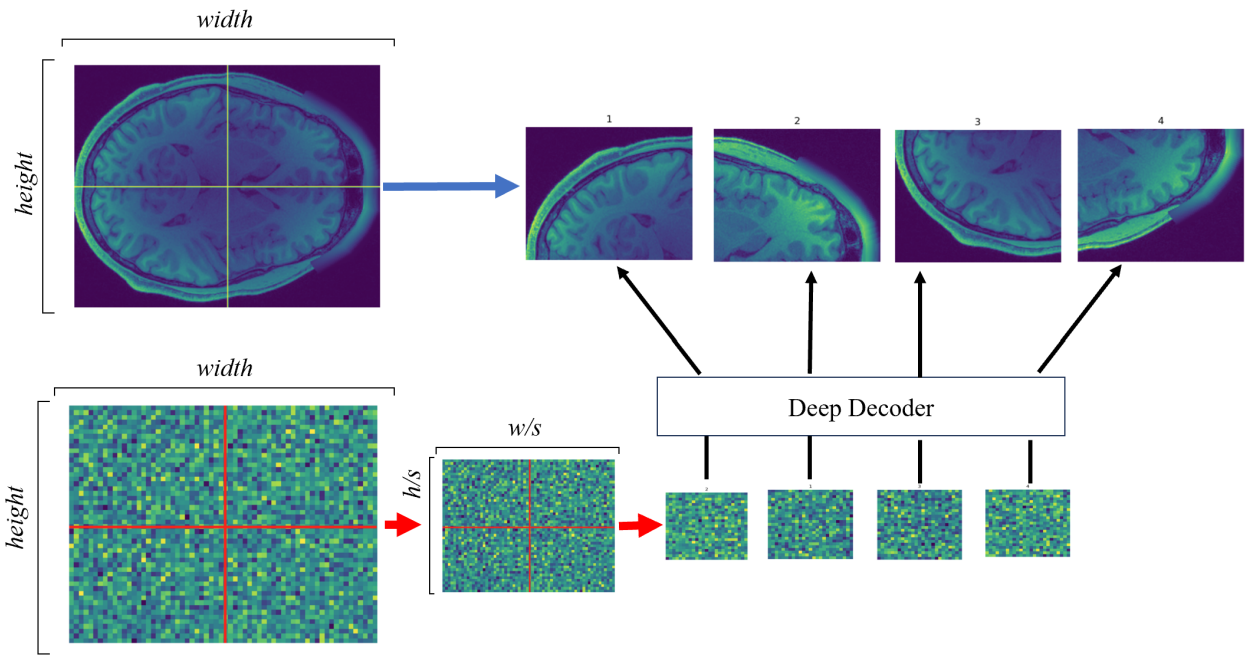


Figure 3.5: Fitting process of the DD using patch-noise pairs as inputs. The noise is generated with the same size of the target and then scaled by the factor s. Each patch-noise pair corresponds to a specific region of the target image with associated noise.

the start and end tensors. Through linear interpolation between these, we generated a number of noise samples for each of the hidden channels.

For the case of Pelin noise, we created a 3D object of Perlin noise of a certain shape and then divided it into 'slices' to get the hidden channel corresponding samples.

3.3 Grid Search

After setting the inputs and conducting several qualitative experiments, we proceeded to fit the Deep Decoder (DD) for all potential combinations of noise and target images, encompassing the three types of noise explained in Section 3.1 and the target options detailed in Section 3.2. Subsequently, we obtained the output images and compared them with the original mid slice to assess the decoder's ability to reproduce image details.

For each of the 12 options available, a grid search was performed to optimize the model's hyperparameters, including the number of channels **k**, the size of random inputs to be upsampled (determined by dividing by a scale factor **s**), the number of blocks of consecutive layers **l**, and the number of fitting iterations. This comprehensive search resulted in a total of 189 combinations (See Table 3.1).

The grid search allowed us to fine-tune the model's parameters, ensuring it produced the most accurate and representative results across diverse scenarios. This systematic exploration paved the way for a robust evaluation of the Deep Decoder's performance in various noise and target image conditions.

Moreover, [GSI] demonstrated that an ensemble of DIP models, in that case, aided in achieving improved output. Inspired by this, we also perform a grid search by initializing five DD models

k	s	l	# iterations
$\{8, 16, 32, 64, 96, 128, 192\}$	$\{2^5, 2^4, 2^3\}$	$\{4, 5, 6\}$	$\{2000, 5000, 7000\}$

Table 3.1: Hyperparameters options for the grid search.

simultaneously and computing their average output to assess potential improvements in the results (albeit at the cost of increased computation time):

$$\hat{x} = \frac{1}{n} \sum_{i=1}^{n} \hat{x_i} \tag{3.3}$$

In this equation, $\hat{x}$ represents the average output obtained by combining the results of n DD outputs ($\hat{x_i}$), 5 in this case. By taking the average of the individual model outputs, we seek to explore the potential benefits of an ensemble approach in enhancing the final output quality in exchange of a longer computation time.

3.4 Metrics

The following metrics were used to compare the output of the DD with the ground truth, implemented with the piq (PyTorch Image Quality) library in Python [KZPD22], as they are the common standard for measuring output quality similar lines of research [YJG+, ZDH].

3.4.1 PSNR

Peak signal-to-noise ratio (PSNR) is the most commonly used metric to asses for image quality, measured in decibels (dB). It takes into account the ration between the maximum value of the image and the mean squared error (MSE):

$$\text{PSNR} = -10 \cdot \log_{10} \left(\frac{\max(\hat{x})^2}{\text{MSE}} \right) \tag{3.4}$$

where:

- $\max(\hat{x})$ represents the maximum value in the processed image $\hat{x}$, i.e., the maximum pixel intensity value present in the image.
- MSE is the Mean Squared Error between the processed and reference images, calculated as the average of the squared differences between corresponding pixels in the two images:

$$\text{MSE}(\hat{x}, y) = \frac{1}{m \cdot n} \sum_{i=1}^{m} \sum_{j=1}^{n} (\hat{x}(i,j) - y(i,j))^2 \tag{3.5}$$

  Where $\hat{x}$ represent the output image, and y represents the reference (ground truth) image. The images have dimensions m (height or number of rows) and n (width or number of columns).

The PSNR measures the distortion between the processed and reference images. A higher PSNR value indicates that the images are more similar (usually, a 'good PSNR' is around 30dB), and the processed image has less distortion compared to the reference image. Since the images are normalized between [0,1], $max(\hat{x}^2) = 1$ and therefore the formula can be simplified as follows:

$$\text{PSNR} = 10 \cdot \log_{10} (\text{MSE}) \tag{3.6}$$

3.4.2 SSIM and MS-SSIM

SSIM (Structural Similarity Measure) [WBSS] between two provided images. Instead of considering absolute differences in the images such as with the MSE, it tries to give a metric based on the structural differences between the images with values between [-1,1], being 1 completely equal and -1 completely different images. The assumption is that the human visual perception system possesses remarkable ability in discerning structural details within a scene, allowing it to perceive distinctions between information extracted from a reference and a sample image. Consequently, a metric that emulates this behavior would demonstrate superior performance in tasks requiring discrimination between sample and reference images. It takes into account the luminance $\boldsymbol{l}$, contrast $\boldsymbol{c}$ and structure $\boldsymbol{s}$ between the two images:

$$\text{SSIM}(\hat{\mathbf{x}}, \mathbf{y}) = [l(\hat{\mathbf{x}}, \mathbf{y})]^{\alpha} \cdot [c(\hat{\mathbf{x}}, \mathbf{y})]^{\beta} \cdot [s(\hat{\mathbf{x}}, \mathbf{y})]^{\gamma} \tag{3.7}$$

With α, β, γ, being the importance of each parameter in the metric.

- For the luminance comparison (measured by averaging the pixel values of each image) we use the following :

$$l_{\hat{x},y} = \frac{2\mu_{\hat{x}}\mu_y + C_1}{\mu_{\hat{x}}^2 + \mu_y^2 + C_1} \tag{3.8}$$

  with μ being the average intensity of each image and C1 being a constant to avoid instability:

$$\mu_{\hat{x}} = \frac{1}{m \cdot n} \sum_{i=1}^{m} \sum_{j=1}^{n} \hat{x}(i,j) \qquad \mu_y = \frac{1}{m \cdot n} \sum_{i=1}^{m} \sum_{j=1}^{n} y(i,j) \tag{3.9}$$

- For the contrast comparison (measured by considering the standard deviation σ) :

$$c(\hat{\mathbf{x}}, \mathbf{y}) = \frac{2\sigma_{\hat{x}}\sigma_y + C_2}{\sigma_{\hat{x}}^2 + \sigma_y^2 + C_2} \tag{3.10}$$

  with σ being the standard deviation of the intensity values in the image and C2 another constant for ensure stability:

$$\sigma_{\hat{x}} = \sqrt{\frac{1}{(m \cdot n) - 1} \sum_{i=1}^{m} \sum_{j=1}^{n} (\hat{x}(i,j) - \mu_{\hat{x}})^2} \quad \sigma_y = \sqrt{\frac{1}{(m \cdot n) - 1} \sum_{i=1}^{m} \sum_{j=1}^{n} (y(i,j) - \mu_y)^2} \tag{3.11}$$

- Finally for the structure comparison:

$$s(\hat{\mathbf{x}}, \mathbf{y}) = \frac{\sigma_{\hat{x}y} + C_3}{\sigma_{\hat{x}}\sigma_y + C_3} \tag{3.12}$$

  Where $\sigma_{\hat{x}y}$ is the covariance of the pixel values of the images:

$$\sigma_{xy} = \frac{1}{(m \cdot n - 1)} \sum_{i=1}^{m} \sum_{j=1}^{n} (\hat{x}(i,j) - \mu_{\hat{x}}) \cdot (y(i,j) - \mu_y) \tag{3.13}$$

By combining all the given parameters and setting $\alpha = \beta = \gamma = 1$ and $C3 = C2/2$ as in the work by Wang et al [WSB], the resulting expression is as follows:

$$\text{SSIM}(\hat{\mathbf{x}}, \mathbf{y}) = [l(\hat{\mathbf{x}}, \mathbf{y})]^{\alpha} \cdot [c(\hat{\mathbf{x}}, \mathbf{y})]^{\beta} \cdot [s(\hat{\mathbf{x}}, \mathbf{y})]^{\gamma} = \frac{(2\mu_{\hat{x}}\mu_y + C_1)\,(2\sigma_{\hat{x}y} + C_2)}{\left(\mu_{\hat{x}}^2 + \mu_y^2 + C_1\right)\left(\sigma_{\hat{x}}^2 + \sigma_y^2 + C_2\right)} \tag{3.14}$$

Moreover [WSB] introduced the concept of MS-SSIM or Multi Scale SSIM, which is a similar metric, but considering the SSIM at different L scales i of the image in order to account not only for global differences but also locally. It downsamples the images along with applyting a Gaussian kernel and weights $\{w_1, w_2, \ldots, w_L\}$ each scale in order to compute the SSIM, capturing the relevance of both fine details and larger structures:

$$\text{MS-SSIM}(x, y) = \prod_{i=1}^{L} (\text{SSIM}_i(\hat{x}, y))^{w_i} \tag{3.15}$$

Unfortunately, the computation of the MS-SSIM was not possible in the patch-based case due to the small size of the image patches, making it unable to perform the downsampling of the images.

3.4.3 VIF

Visual Information Fidelity (VIF) [SB] is used for assessing image quality based on the Human Visual System with typical values between [0,1], but can be higher for output images with higher contrast than the ground truth. While not as widely employed as some other scores, VIF has proven to be highly valuable in certain contexts such as radiology [MRC+]. This score evaluates the extent to which an output image preserves visual information when compared to a ground truth or reference image. In essence, it measures how well the important visual details are retained in the distorted or processed image. Similarly to the MS-SSIM, it is calculated for several scales s of the image (4 scales in our case). It is calculated as follows:

- For each scale i, $\text{VIF}_{\text{scale}}(\hat{x}, y)_i$ is computed using the formula:

$$\text{VIF}_{\text{scale}}(x, y)_i = \frac{\log_{10}\left(1 + (g_i^2) \cdot \frac{\sigma_y^2}{\sigma_v^2 + \sigma_n^2}\right)}{\log_{10}\left(1 + \frac{\sigma_y^2}{\sigma_n^2}\right)} \tag{3.16}$$

  where g_i is the gain factor at each scale, σ_y^2 is the local variance of the downsampled ground truth image y, σ_v^2 is the local variance of the adjusted distorted image $\hat{x}$ at scale i, and σ_n^2 is the variance of the visual noise (a pre-fixed value of 2 in this case):

$$g = \frac{\sigma_{xy}}{\sigma_y^2} \tag{3.17}$$

$$\sigma_v^2 = \sigma_x^2 - g_i \cdot \sigma_{xy} \tag{3.18}$$

- After obtaining the VIF values for all scales i, they are aggregated to compute the final VIF index using the formula:

$$\text{VIF}(x, y) = \frac{\sum_{i=0}^{s} \text{VIF}_{\text{scale}}(x, y)_i}{\sum_{i=0}^{s} \log_{10}\left(1 + \frac{\sigma_y^2}{\sigma_n^2}\right)} \tag{3.19}$$

The final value of $\text{VIF}(x, y)$ represents the Visual Information Fidelity index between the two input images x and y.

Chapter 4

Results

After the grid search, some samples of the outputs images can be seen in Figure 4.1.

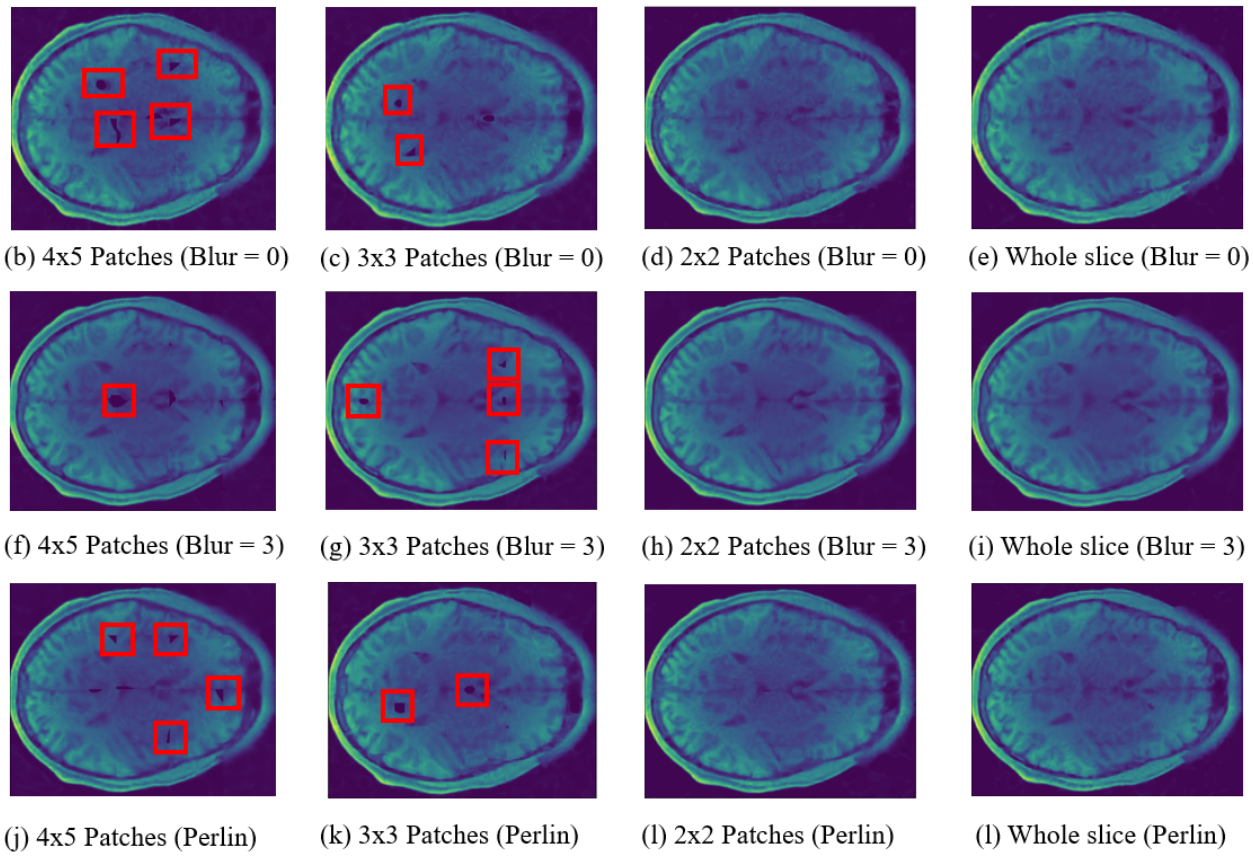


Figure 4.1: Output variations with diverse input noises (Blur=0, Blur=3 and Perlin Noise) and target image types (5x4 Patches, 3x3 Patches, 2x2 Patches, and Whole Slice). Artifacts are shown in the red bounding boxes. In this case, all the hyperparameters were set equally to allow comparison, namely $k = 64$, $s = 2^4$, $l = 4$ and 2000 iterations.

Upon initial observation, it becomes evident that when too many target images (along with fixed random inputs) are fed to the DD, artifacts appear, mainly in areas with slight variations of intensity, typically at the edges between patches. These artifacts are only present when using 5x4

and 3x3 patches. However, when 2x2 patches or the entire slice are presented to the model, these artifacts vanish.

Additionally, the use of a Gaussian kernel in the input noise noticeably improves the metrics compared to no blur (this can be observed qualitatively in Figure 4.1 second row compared to the first row), resulting in a higher-quality output image. However, it should be noted that the Gaussian noise seems to affect the fine details of the image, making it suitable for recovering the overall shape of the slice but less effective in preserving smaller structures' details. In contrast, the application of Perlin noise slightly avoids this issue by maintaining a few more fine details at the cost of some image quality, representing a middle ground between the presence and absence of blurred noise as an input.

In general (See Section 4.2 on Stability of the DD), by employing multiple initializations of the DD simultaneously, we observed a proportional increase in processing time. However, this approach only yielded a slight improvement in PSNR. Nevertheless, despite the cost of ≈ 1 or 2dB, the pursuit of an optimal combination of hyperparameters proved to be worthwhile.

Based on the quantitative examples provided, which offer a representation of the overall behavior of the DD concerning the given inputs and targets, our next step involves analyzing the data obtained from the grid search. The aim is to gain insights into how hyperparameters impact performance and inference time. Notably, unlike trained models that execute a forward pass, the DD follows an iterative approach to solve the optimization problem.

4.1 Inference time

Firstly, we examine how the hyperparameters affect the inference time, considering trivial the fact that a larger number of iterations increases the computation time. In Figure 4.2, we present the average inference time for the 189 combinations of hyperparameters across the 12 possible setups of input/target pairs.

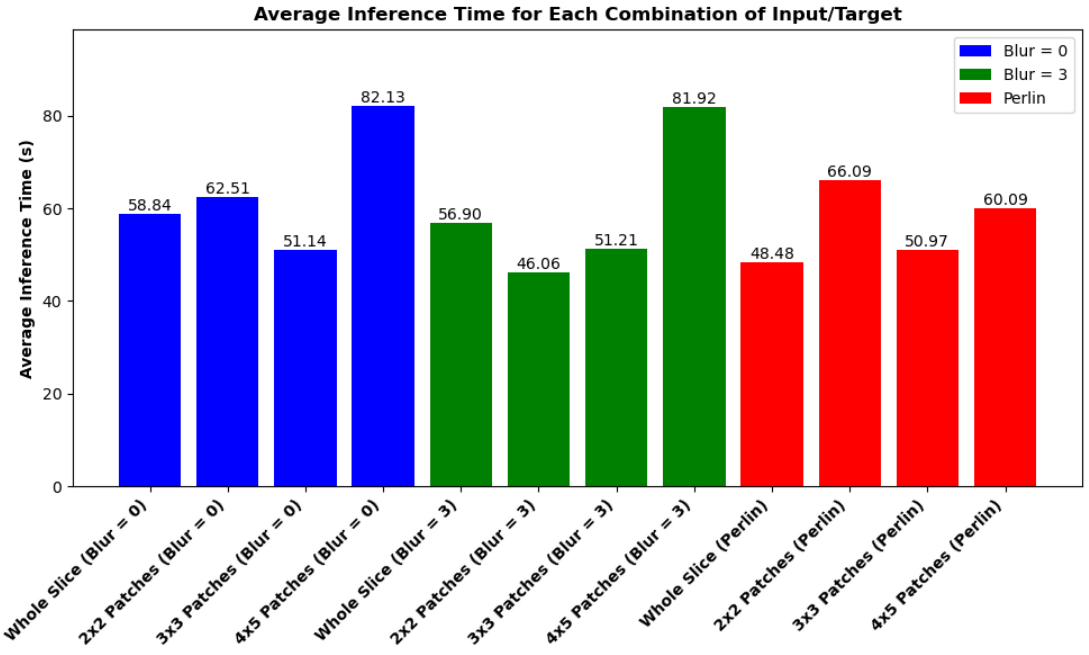


Figure 4.2: Average Inference Time for Different Input/Target Combinations.

From the plot, we can first appreciate that the inference time in the cases of Blur = 0 and Blur = 3 has a similar behavior, especially when the input/target pairs are 4x5 patches (20 images in total) for which the time increases notably. This is not seen when the fixed input is Perlin noise, suggesting that the inference time is dependent on an interaction between the input size and its nature, considering that the other hyperparameters remain fixed. Nevertheless, averaging the inference time by the type of noise, we find that for Blur = 0 it is 63.7 *seconds*, for Blur = 3 it is 59.0 *s* and in the case of Perlin we obtain 56.4 *s*. On the other hand, averaging by the number of patches we find that for the Whole Slice inference time is 54.7 *s*, for 2x2 Patches 58.2 *s*, for 3x3 Patches 51.1 *s* and for 4x5 Patches 74.4 *s* showing similar values except in the case for 4x5 Patches:

Noise Type	**Average Inference Time (s)**
Blur = 0	63.7
Blur = 3	59.0
Perlin	56.4

Input Type	**Average Inference Time (s)**
Whole Slice	54.7
2x2 Patches	58.2
3x3 Patches	51.1
4x5 Patches	74.4

Table 4.1: Inference Time by Noise Type and Patch Size

Moreover, all these times are smaller on average than the original implementation of the DD [HH19] (can be found at `https://github.com/reinhardh/supplement_deep_decoder`), in which the number of iterations is set to 20000 but yields similar performance (This motivates Section 4.2 on Stability).

When solely considering the number of hidden channels $\boldsymbol{k}$, it becomes evident that augmenting this value leads to an increase in processing time, particularly noticeable when exceeding 64 channels (See Figure 4.3).

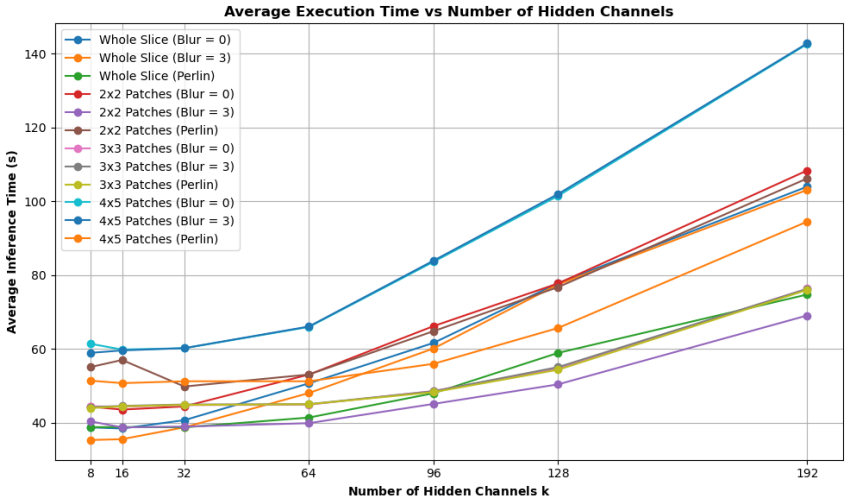


Figure 4.3: Average Inference Time vs Number of Hidden Channels $\boldsymbol{k}$. From 8 to 64 the time seems similar, but it highly increases afterwards.

Interestingly, the collected data revealed a consistent trend across all cases, indicating that both increasing the number of blocks $\boldsymbol{l}$ and using a smaller scaling factor $\boldsymbol{s}$ (resulting in larger input noise to be upsampled) led to heightened processing time. To avoid overwhelming the reader with extensive data, Table 4.2 showcases this phenomenon for the case of 4x5 Patches and Blur = 0, providing a representative sample that is applicable to all input/target pairs.

Moreover, the data also suggests that the rise in inference time is more influenced by the number of layers rather than the scaling factor. This observation could be attributed to the increased presence of convolution layers as the number of blocks $\boldsymbol{l}$ is augmented.

$\boldsymbol{l}$	$\boldsymbol{s}$	**Inference time (s)**
4	2^5	28.645849
4	2^4	28.937979
4	2^3	30.816761
5	2^5	33.452213
5	2^4	35.364100
5	2^3	37.517176
6	2^5	39.260080
6	2^4	41.365572
6	2^3	43.957175

Table 4.2: Inference Time by Blocks of Layers $\boldsymbol{l}$ and scaling factor $\boldsymbol{s}$ for 2000 iterations and 8 channels.

4.2 Stability of the DD

A noteworthy aspect of the DD in this project was its occasional production of highly distorted output images that deviated significantly from the expected results. However this was usually solved by running the model again. While certain configurations yielded reasonable results every time, there were instances where the DD produced random outcomes, underscoring the need for careful hyperparameter tuning to achieve consistent and satisfactory results.

To get insights of this behaviour, we take advantage of the 5 parallel initializations done of the DD. The PSNR of each of the initialization outputs was saved and their means and variances were obtained, hence, the coefficient of variation (CV) could be calculated (more robust that simply identifying the settings with high PSNR variance). The CV allows us to standardize the comparison of relative variability, making it easier to discern the consistency and stability of the results across different initializations. For the 5 PSNR_i values is calculated as follows:

$$\text{CV } (\%) = \left(\frac{\sqrt{\sigma^2}}{\mu} \right) \times 100 \tag{4.1}$$

where μ is the mean given by $\mu = \frac{\sum_{i=1}^{5} \text{PSNR}_i}{5}$ and σ^2 represents the variance, computed as $\sigma^2 = \frac{\sum_{i=1}^{5} (\text{PSNR}_i - \mu)^2}{5}$.

Figure 4.4 shows the percentage of combinations out of the 189 possible settings in which the CV has larger than 5% which we set as the threshold for high variance, since we expect consistent results across initializations.

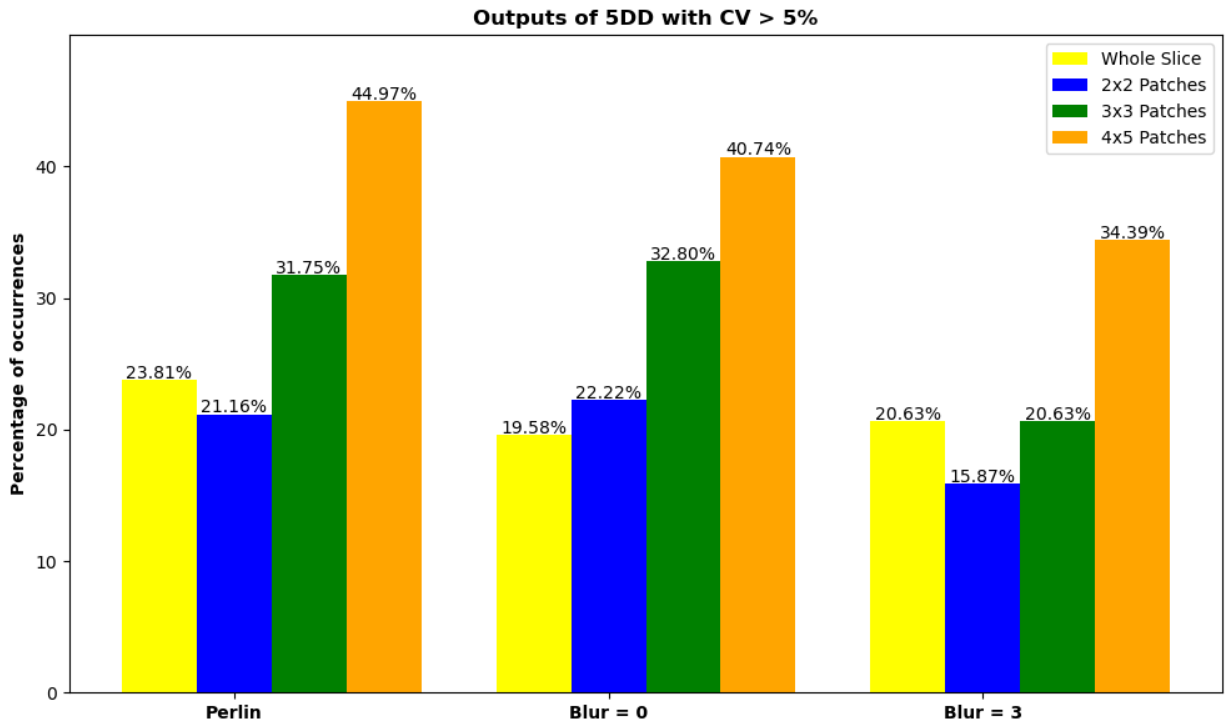


Figure 4.4: For each input/target pair, the figure shows the percentage of combinations for which the outputs vary in different initializations with CV $> 5\%$.

Interestingly, several trends can be observed in the plot. Firstly, it is noteworthy that the percentage of variations are similar in cases where Perlin Noise and Uniformly Distributed Noise (Blur $= 0$) are used, leading to a larger number of instances where this phenomenon occurs compared to cases with Blur $= 0$ alone. Additionally, one can observe an increase in occurrences when using 3x3 and 4x5 Patches (green and orange bars), suggesting that a larger batch size fed to the decoder, i.e., an increased number of total patches, contributes to the model's instability. This observation aligns with a lower instability noticed when fewer images are fed to the DD, as seen in the case of Whole Slice and 2x2 Patches.

Upon conducting a more in-depth exploration of the data, a significant observation emerged: the majority of cases across all 12 input/target pairs exhibited high CV primarily when the number of channels increased, specifically in scenarios where more than 64 hidden channels were employed. Notably, there were no instances of high CV values when 64 hidden channels or fewer were used. However, no other clear tendency was identified with the rest of the hyperparameters.

To illustrate these instabilities, we show in Figure 4.5 an example of two extreme case settings, one with a very high CV and the another with a low CV among initializations. The figure displays single outputs obtained from random initializations for each of these settings. In this particular case, we consider Whole Slice and Blur $= 3$ as the chosen configuration[1].

[1]Bear in mind that the examples showcased here are representative of the findings obtained after an exhaustive exploration of all possible settings and combinations of input/target pairs. However, we present only a subset of these examples for the sake of clarity and conciseness.

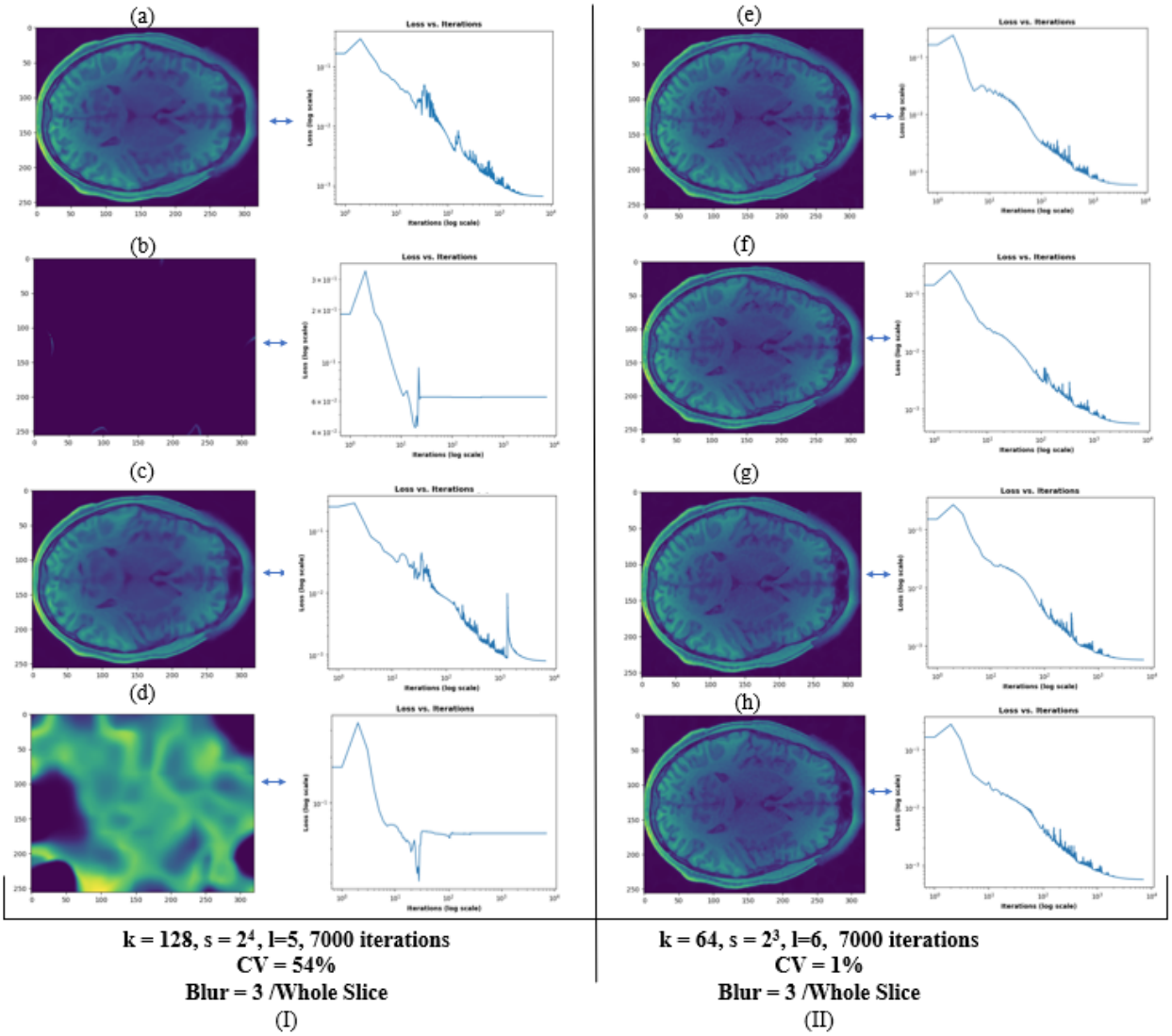


Figure 4.5: On the left (I) an extreme setting in which different initializations lead to completely different results vs right (II) in which the results are similar among initializations and convergence curves are alike.

The convergence curves depicted in Figure 4.5 showcase the MSE loss plotted against the iteration count during the model's optimization process. Figures I-c and I-d reveal fatal instances where the optimizer gets stuck in a local minimum, preventing it from escaping, thus indicating the possibility of further improvements through the use of alternative optimizers or loss functions. However, not only do these results appear to be unstable, but they also consume more time, indicating that setting the channel number to values over 128 should be avoided to mitigate this issue. Interestingly, in the stable cases, the model exhibits a homogeneous behavior of the convergence curves. Also remarkably, the scores for the VIF and SSIM in the unstable instances turned out to be unrealistic. This might be due to the outputs not fulfilling the assumptions made by the implementation of the library for calculating these metrics. As a result, it is suggested to rely on visual inspection to identify such fatal cases or trust the PSNR scores for discrimination[2].

[2]MS-SSIM values were always coherent with the fatal or correct results, however, this was only implemented for

4.3 Outputs quality

After identifying and discarding the settings in which the CV was larger than 5% or in which there were poor (PSNR < 15) or unrealistic scores (VIF > 1) we found out that for every input/target pair, the settings in which $k = \{128, 196\}$ were discarded applying these conditions, along with a minority of specific different cases for each pair. Figure 4.6 shows the averaged PSNR values for the stable settings of each combination, scoring the output of a single random initialization of a DD (dark coloured, left bars) and the average PSNR of 5DD (light coloured, right bars):

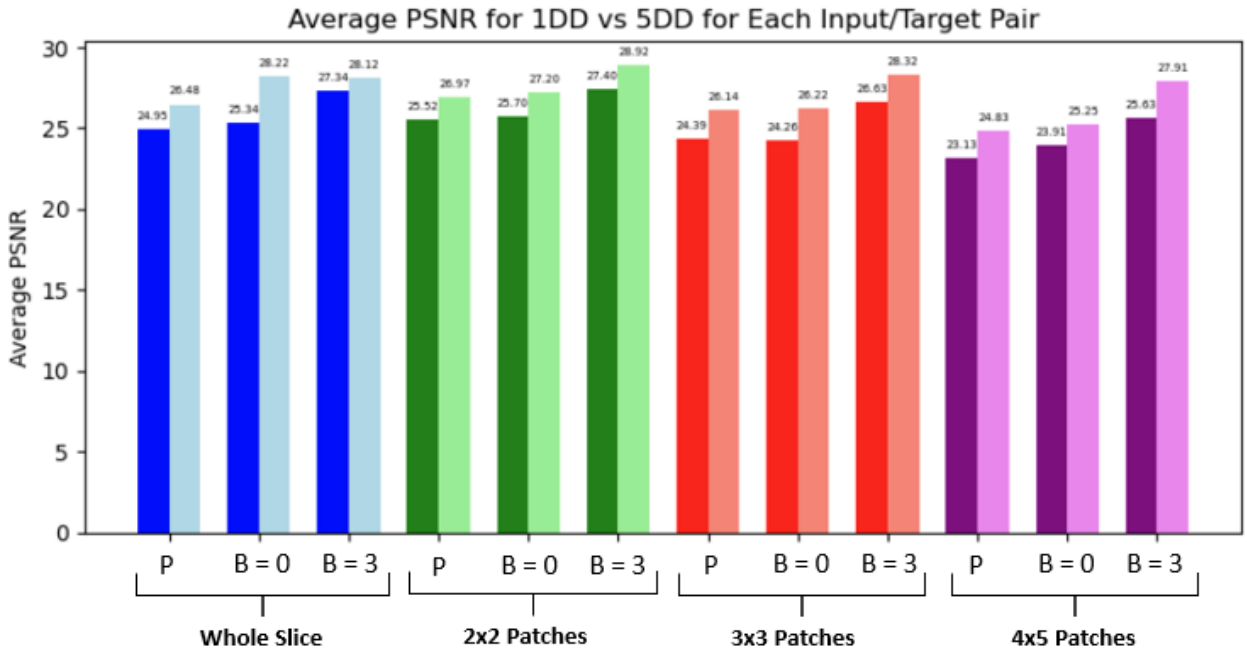


Figure 4.6: Average PSNR for 1DD vs 5DD for Each Input/Target Pair. Grouped by target type and then divided by input noise type (P being Perlin, B = 0 being Blur = 0 and B = 3 being Blur = 3). In all cases, PSNR increases with more initializations averaged rather than a single one.

It is worth mentioning that the observed improvements are on the order of approximately 2 to 3dBs, which can be valuable for very specific tasks. However this comes at the cost of a proportional increase in inference time, hence 5 initializations are equivalent to 5 times larger inference time, which might not be useful in the case of automatic tasks with a large number of images. Data also suggests that, in the case of scoring the outputs by PSNR value, using blurry noise improves the result. However, we must look at the other metrics to asses the quality of the outputs.

When searching for the best sets of hyperparameters based on the scores[3], we observed an interesting trend. In most cases, when identifying the settings that achieved the highest PSNR and SSIM scores, these two metrics were highest in the same configurations. However, it was not always the case for VIF. Despite this, when focusing on the settings that yielded the highest VIF scores, we still found that the PSNR and SSIM scores were also notably high. Hence, while PSNR and SSIM demonstrated a consistent correlation, VIF exhibited some variation in its correlation with the other metrics.

the Whole Slice cases as explained in Section 3.4.2

[3]Note that this analysis was conducted for the 5 initialization cases to ensure robustness, even though filtering was already performed. Similar results can be expected for single initializations.

Also we discovered that even in the settings with high metrics, for the cases of 4x5 Patches and 3x3 the 'black wholes' artifacts showed at the beginning in Figure 4.1 remained although they were less notable. In Tables 4.3, 4.4 and 4.5 we show the best hyperparameter settings for each input/target pair between curly brackets next to the metric value.

	Settings $\{k, s, l, \#$ iterations$\}$ **and best PSNR Values**		
	Perlin	**Blur = 0**	**Blur = 3**
Whole	{96, 2^4, 6, 7000}, 32.77	{96, 2^3, 6, 7000}, 33.55	{96, 2^3, 6, 5000}, 33.84
2x2 Patches	{96, 2^3, 6, 5000}, 32.56	{64, 2^4, 6, 7000}, 32.63	{96, 2^3, 6, 7000}, 33.35
3x3 Patches	{96, 2^3, 5, 7000}, 32.13	{64,2^3, 6, 7000}, 32.26	{96, 2^3, 5, 5000}, 33.56
4x5 Patches	{96, 2^4, 5, 5000}, 30.78	{96, 2^3, 5, 7000}, 31.32	{96, 2^3, 4, 7000}, 32.73

Table 4.3: Settings with best PSNR Values

	Settings $\{k, s, l, \#$ iterations$\}$ **and best SSIM Values**		
	Perlin	**Blur = 0**	**Blur = 3**
Whole	{96, 2^4, 6, 7000}, 0.88	{96, 2^3, 6, 7000}, 0.90	{96, 2^3, 6, 5000}, 0.91
2x2 Patches	{64, 2^3, 6, 7000}, 0.88	{96, 2^3, 6, 5000}, 0.88	{64, 2^3, 6, 7000}, 0.89
3x3 Patches	{96, 2^3, 5, 7000}, 0.86	{64, 2^3, 6, 7000}, 0.87	{96, 2^3, 5, 5000}, 0.90
4x5 Patches	{96, 2^4, 5, 5000}, 0.82	{96, 2^3, 5, 7000}, 0.84	{96, 2^3, 4, 7000}, 0.87

Table 4.4: Settings with best SSIM Values

	Settings $\{k, s, l, \#$ iterations$\}$ **and best VIF Values**		
	Perlin	**Blur = 0**	**Blur = 3**
Whole	{96, 2^4, 6, 7000}, 0.42	{96, 2^3, 4, 5000}, 0.45	{96, 2^4, 6, 7000}, 0.46
2x2 Patches	{96,2^3, 6, 5000}, 0.42	{64, 2^4, 6, 7000}, 0.42	{64, 2^3, 6, 7000}, 0.45
3x3 Patches	{96, 2^3, 5, 7000}, 0.36	{64, 2^3, 6, 7000}, 0.37	{96, 2^3, 5, 5000}, 0.42
4x5 Patches	{96, 2^4, 5, 5000}, 0.24	{96, 2^3, 5, 7000}, 0.26	{96, 2^3, 5, 7000}, 0.33

Table 4.5: Settings with best VIF Values

To avoid overwhelming the reader with an abundance of numbers, let's summarize the main findings. As Figure 4.6 previously suggests, we observed that as the batch size (and consequently, the number of patches) increased, not only did the model's instability grow, as previously shown, but it also performed poorly even in the best-case scenarios. This decline in performance could be attributed to the persistent presence of 'black hole' artifacts.

Additionally, we noticed that the typical best configurations did never involve 2000 iterations, a scaling factor of 2^5 or less than 64 hidden channels, nor 4 consecutive layers. This observation was consistent even when considering the top five best configurations for each combination.

Hence one could state that the minimum requirements for a good performance would be 65 hidden channels, scaling the input by a factor of not larger than 2^4, not less than 5 blocks of consecutive layers and not less than 5000 iterations. If we wanted to give an upper bound, more than 96 hidden channels seems to increase the instability of the model giving fatal results,

Chapter 5

Discussion and future outlook

Once we analyzed the results after an exhaustive search to comprehend how hyperparameters affect the behaviour of the DD we could state some minimum requirements for a good performance. These start by using not less than 65 hidden channels (preferably 96), scaling the input by a factor of not larger than 2^4, not less than 5 blocks of consecutive layers and not less than 5000 iterations. Conversely, if we consider an upper bound, employing more than 96 hidden channels appears to increase the model's instability, resulting in fatal outcomes and significantly impacting the inference time and not more than 7000-10000 should be necessary to have a good performance. Our findings suggest that the number of hidden channels play a mayor role in the performance of the model. We mention also that the use of Gaussian blur seems to be the most effective initialization of the fixed input noise, however, perlin noise still needs to be explored as it comprises many different options when tunning their feaure scales (in Section3.1.2). Furthermore, these findings seem to be consistent when experimenting with MRI mid slices coming from different subjects. However, as a future research line, preliminary experiments done by fixing a hyperparameter setting to resolve a 3D MRI volume did not have a good performance (See Figure 5.1), since the further from the mid slice, the worse the reconstruction of the ground truth was.

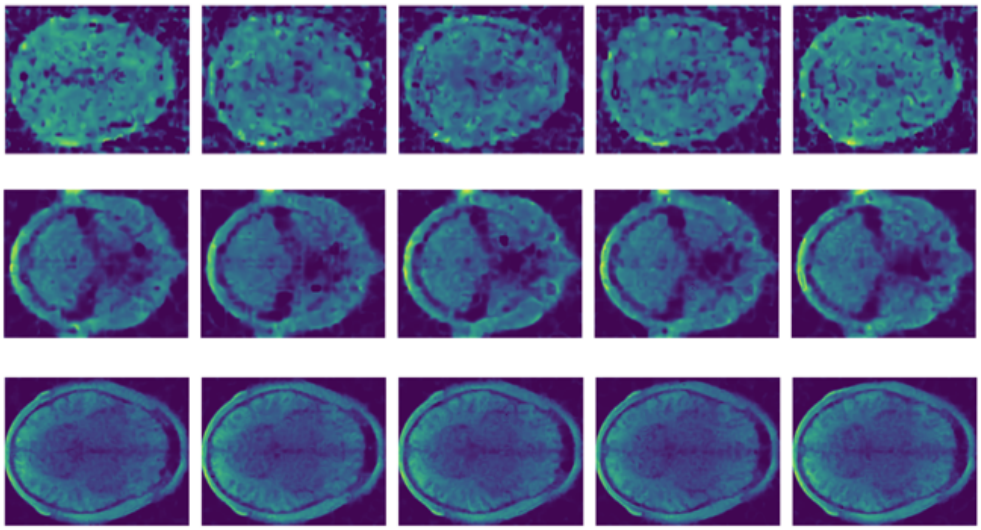


Figure 5.1: Experiments with setting hyperparameters for many different slices. The results are worse when going further from the mid slice for which the hyperparemters were set

This might suggest that I) The hyperparameter choice is highly related to the complexity and geometry of the target images and 2) That the model finds it challenging to fit for several images at once (as it happened with the artifacts present in the 3x3 and 4x5 Patches case) since larger batch sizes are more unstable and difficult to optimize. One idea for fitting the model with several images of similar geometry and complexity (this is not the case in patches since their geometries are quite different) could be based on initializing the DD already fitted for a previous image, however we also proved that when many images were to be fitted, the inference time increased notably (See Table 4.1). Experiments of this kind by initializing the biases of the layers in the model did not change the results in our 2D experiments. Also, the scalability of the DD for automatic tasks still seems challenging due to the larger inference time required in comparison with trained neural networks.

Finally we remark that our experiments shed light on developing an intuition on how DD can behave by tuning its inputs, targets and hyperparameters in an exhaustive way.

*The scripts used during the development of this project can be found in `https://gitlab.ethz.ch/pblasco/deepdecoder-semester-project/`

Bibliography

[BOS] George Barbastathis, Aydogan Ozcan, and Guohai Situ. On the use of deep learning for computational imaging. 6(8):921.

[Bur] Wilhelm Burger. *Gradientenbasierte Rauschfunktionen Und Perlin Noise*.

[DBP+] Magnus Dustler, Predrag Bakic, Hannie Petersson, Pontus Timberg, Anders Tingberg, and Sophia Zackrisson. Application of the fractal Perlin noise algorithm for the generation of simulated breast tissue. 9412.

[DLHT16] Chao Dong, Chen Change Loy, Kaiming He, and Xiaoou Tang. Image Super-Resolution Using Deep Convolutional Networks. *IEEE transactions on pattern analysis and machine intelligence*, 38(2):295–307, February 2016.

[DSHG+21] August DuMont Schütte, Jürgen Hetzel, Sergios Gatidis, Tobias Hepp, Benedikt Dietz, Stefan Bauer, and Patrick Schwab. Overcoming barriers to data sharing with medical image generation: a comprehensive evaluation. *npj Digital Medicine*, 4(1):1–14, September 2021. Number: 1 Publisher: Nature Publishing Group.

[EHN] Heinz W. Engl, Martin Hanke, and Andreas Neubauer. *Regularization of Inverse Problems*. Number v. 375 in Mathematics and Its Applications. Kluwer Academic Publishers.

[GSI] Yosef Gandelsman, Assaf Shocher, and Michal Irani. "Double-DIP": Unsupervised Image Decomposition via Coupled Deep-Image-Priors. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 11018–11027. IEEE.

[HH19] Reinhard Heckel and Paul Hand. Deep Decoder: Concise Image Representations from Untrained Non-convolutional Networks, February 2019. arXiv:1810.03982 [cs, stat].

[HZL+] Zhuonan He, Jinjie Zhou, Dong Liang, Yuhao Wang, and Qiegen Liu. Learning Priors in High-frequency Domain for Inverse Imaging Reconstruction.

[JH] Gauri Jagatap and Chinmay Hegde. High Dynamic Range Imaging Using Deep Image Priors. In *ICASSP 2020 - 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 9289–9293. IEEE.

[KZPD22] Sergey Kastryulin, Jamil Zakirov, Denis Prokopenko, and Dmitry V. Dylov. Pytorch image quality: Metrics for image quality assessment, 2022.

[LGG+] Brendon Lutnick, Brandon Ginley, Darshana Govind, Sean D. McGarry, Peter S. LaViolette, Rabi Yacoub, Sanjay Jain, John E. Tomaszewski, Kuang-Yu Jen, and Pinaki Sarder. An integrated iterative annotation technique for easing neural network training in medical image analysis. 1(2):112–119.

[LSXK] Jiaming Liu, Yu Sun, Xiaojian Xu, and Ulugbek S. Kamilov. Image Restoration using Total Variation Regularized Deep Image Prior.

[LTMK] Santiago López-Tapia, Rafael Molina, and Aggelos K. Katsaggelos. Deep learning approaches to inverse problems in imaging: Past, present and future. 119:103285.

[MEM] Gary Mataev, Michael Elad, and Peyman Milanfar. DeepRED: Deep Image Prior Powered by RED.

[Mou] Pierre Moulin. Chapter 6 - Multiscale Image Decompositions and Wavelets. In Al Bovik, editor, *The Essential Guide to Image Processing*, pages 123–142. Academic Press.

[MRC+] Allister Mason, James Rioux, Sharon E. Clarke, Andreu Costa, Matthias Schmidt, Valerie Keough, Thien Huynh, and Steven Beyea. Comparison of Objective Image Quality Metrics to Expert Radiologists' Scoring of Diagnostic Quality of MR Images. 39(4):1064–1072.

[OJM+] Gregory Ongie, Ajil Jalal, Christopher A. Metzler, Richard G. Baraniuk, Alexandros G. Dimakis, and Rebecca Willett. Deep Learning Techniques for Inverse Problems in Imaging. 1(1):39–56.

[Per] Ken Perlin. Improving noise. In *Proceedings of the 29th Annual Conference on Computer Graphics and Interactive Techniques*, pages 681–682. ACM.

[Per85] Ken Perlin. An image synthesizer. *SIGGRAPH Comput. Graph.*, 19(3):287–296, jul 1985.

[QIS+] Adnan Qayyum, Inaam Ilahi, Fahad Shamshad, Farid Boussaid, Mohammed Bennamoun, and Junaid Qadir. Untrained Neural Network Priors for Inverse Imaging Problems: A Survey. 45(5):6511–6536.

[QIS+23] Adnan Qayyum, Inaam Ilahi, Fahad Shamshad, Farid Boussaid, Mohammed Bennamoun, and Junaid Qadir. Untrained Neural Network Priors for Inverse Imaging Problems: A Survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 45(5):6511–6536, May 2023. Conference Name: IEEE Transactions on Pattern Analysis and Machine Intelligence.

[QZZX21] Zhen Qin, Qingliang Zeng, Yixin Zong, and Fan Xu. Image inpainting based on deep learning: A review. *Displays*, 69:102028, September 2021.

[SB] H.R. Sheikh and A.C. Bovik. Image information and visual quality. 15(2):430–444.

[SMF] J.-L. Starck, Fionn Murtagh, and Jalal M. Fadili. *Sparse Image and Signal Processing: Wavelets, Curvelets, Morphological Diversity.* Cambridge University Press.

[UVL20] Dmitry Ulyanov, Andrea Vedaldi, and Victor Lempitsky. Deep Image Prior. *International Journal of Computer Vision*, 128(7):1867–1888, July 2020.

[VLL+10] Pascal Vincent, H. Larochelle, Isabelle Lajoie, Yoshua Bengio, and Pierre-Antoine Manzagol. Stacked Denoising Autoencoders: Learning Useful Representations in a Deep Network with a Local Denoising Criterion. *J. Mach. Learn. Res.*, 11:3371–3408, 2010.

[VVJS+] Dave Van Veen, Ajil Jalal, Mahdi Soltanolkotabi, Eric Price, Sriram Vishwanath, and Alexandros G. Dimakis. Compressed Sensing with Deep Image Prior and Learned Regularization.

[WBSS] Z. Wang, A.C. Bovik, H.R. Sheikh, and E.P. Simoncelli. Image Quality Assessment: From Error Visibility to Structural Similarity. 13(4):600–612.

[WBW+20] Fei Wang, Yaoming Bian, Haichao Wang, Meng Lyu, Giancarlo Pedrini, Wolfgang Osten, George Barbastathis, and Guohai Situ. Phase imaging with an untrained neural network. *Light: Science & Applications*, 9(1):77, May 2020.

[WSB] Z. Wang, E.P. Simoncelli, and A.C. Bovik. Multiscale structural similarity for image quality assessment. In *The Thrity-Seventh Asilomar Conference on Signals, Systems & Computers, 2003*, volume 2, pages 1398–1402 Vol.2.

[YJG+] Jaejun Yoo, Kyong Hwan Jin, Harshit Gupta, Jerome Yerly, Matthias Stuber, and Michael Unser. Time-Dependent Deep Image Prior for Dynamic MRI. 40(12):3337–3348.

[ZBH+21] Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht, and Oriol Vinyals. Understanding deep learning (still) requires rethinking generalization. *Communications of the ACM*, 64(3):107–115, March 2021.

[ZDH] Mohammad Zalbagi Darestani and Reinhard Heckel. Accelerated MRI With Un-Trained Neural Networks. 7:724–733.

[ZXY+] Zheng Zhang, Yong Xu, Jian Yang, Xuelong Li, and David Zhang. A survey of sparse representation: Algorithms and applications. 3:490–530.

[ZYB] Zhizhen Zhao, Jong Chul Ye, and Yoram Bresler. Generative Models for Inverse Imaging Problems: From mathematical foundations to physics-driven applications. 40(1):148–163.