UNIVERSITÀ
DEGLI STUDI
DI PADOVA

Università degli Studi di Padova

DIPARTIMENTO DI MATEMATICA "TULLIO LEVI-CIVITA"

Corso di Laurea Magistrale in Matematica

Deep K-SVD for image denoising

Relatore:
Prof. Fabio Marcuzzi

Laureanda: **Maddalena Ardini**
2044677

Correlatore:
Dott. Erik Chinellato

19 Aprile 2024

Anno Accademico 2023/2024

Contents

Nomenclature

$AWGN$ Additive White Gaussian Noise

DCT Discrete Cosinus Transform

DFT Discrete Fourier Transform

$ISTA$ Iterative Soft-Thresholding Algorithm

MAE Mean Absolute Error

MAP Maximum a Posteriori

MLP Multi-Layer Perceptron

$MMSE$ Minimum Mean Squared Error

$MS-SSIM$ Multi-Scale SSIM

MSE Mean Squared Error

NCP Normalized Cumulative Periodogram

$ODCT$ Overcomplete DCT

OMP Orthonormal Matching Pursuit

PDF Probability Density Function

$ReLU$ Rectifier Linear Unit

$SSIM$ Structural Similarity Index Measure

Introduction

Image denoising, which is the problem of removing noise from an image, is one of the most important studied problems in image processing. This is due to its significant applications in various problems including enhancing the clarity of medical images like ultrasound or thermal images, which may suffer from noise interference during sensor acquisition. Additionally, it extends to the improvement of images obtained by spacecraft, which can be compromised by adverse environmental conditions or equipment limitations. Moreover, a more recent everyday application is noise reduction in images captured by digital cameras in smartphones. Noise removal has been utilized for numerous tasks in imaging sciences and recent discoveries have sparked renewed interest in studying and improving this problem. For instance in (7) it was explored whether a denoiser $\mathcal{D} : \mathbb{R}^N \to \mathbb{R}^N$ could solve general inverse problems of the form

$$\mathbf{y} = \mathbf{H}\mathbf{x} + \mathbf{v},$$

where $\mathbf{H}$ is a known matrix, $\mathbf{v}$ is the noise and $\mathbf{y}$ the given measurement vector. Notably, for $\mathbf{H} = \mathbb{I}$ it represents the denoising problem. Such investigations hold potential for various image processing tasks such as deblurring, demosaicing, super-resolution etc., depending on the degradation matrix $\mathbf{H}$. Another intriguing application area is image synthesis, which involves generating natural-looking images without conditioning on any input or initialization. This aspect is also mentioned in (7). Regarding the specific problem of denoising many algorithms have been proposed and improved over the years, each employing different constructions and tools, and more recently, involving Artificial Intelligence. Figure 1 illustrates the annual publication trends concerning image denoising, highlighting the growing interest in the problem. The most studied denoising problem is the removal of zero-mean White Gaussian Noise, while preserving the image content, which will be explored in our thesis. Significant progress has been made in the first 10 years of research, with various algorithms employing diverse approaches. However, interest in further improving results waned. Nonetheless, as will be discussed in the following sections, the best results in terms of noise removal can be achieved mathematically. With the advent of deep learning, as depicted in figure 1 , this has led to the development of new denoisers and improvements to existing ones. In this thesis we will focus on one such algorithms, the Deep K-SVD denoiser introduced in 2021 in

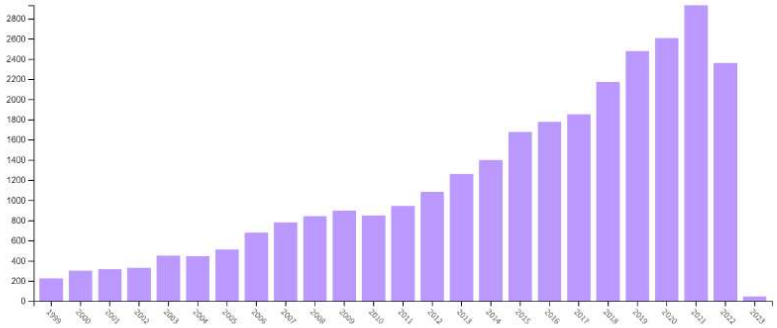

Figure 2.1: The number of papers on the image denoising topic over the years. This graph corresponds to the search topic=((image or video) and (denoising or (noise and remov) or clean)) performed on December 1st 2022 in Clarivate Web-of-Science (WoS). Note that the lower count in 2022 does not stand for a new trend, but rather caused by a delayed reporting of new papers.

Figure 1

(1). This algorithm is the deep version of a classical denoiser that applies the K-SVD algorithm to image denoising, first proposed in 2006 in (2). We will theoretically describe the algorithm in chapter 1, delve into the deep architecture in chapter 2, and study the theoretical considerations for implementing the architecture. For implementation, we considered the original Python code (3), translating all structures using only the `NumPy` library to enhance understanding of the architecture, particularly regarding training improvements and algorithm performance enhancements. We focused on the training loss function, determining how to choose the optimal regularization parameter of a combination of two different functions. This have been done using a Normalized Cumulative Periodogram, which is a technique introduced by Hansen a few years ago in (14) and we apply it here in a novel context of accuracy for a perception-distortion tradeoff. Finally, in Chapter 3, we showcase some of the results obtained.

Chapter 1

Image Denoising

1.1 Problem Definition

Image denoising is an ill-posed problem, meaning that there not exist an unique solution for it. The general problem is that given a clean image, $\mathbf{X} \in \mathbb{R}^N$, assumed to be in in the grayscale range for simplicity, we measure its noisy version $\mathbf{Y} \in \mathbb{R}^N$, given by

$$\mathbf{Y} = \mathbf{X} + \mathbf{V}. \tag{1.1}$$

Here $\mathbf{V} \in \mathbb{R}^N$ is a zero-mean white Gaussian noise, $\mathbf{V} \sim \mathcal{N}(0, \sigma^2\mathbf{I})$, randomly computed. A noise signal, in this case, an image, is called white noise when it satisfies

$$\mathbb{E}(\mathbf{V}) = 0 \quad \text{and} \quad \text{Cov}(\mathbf{V}) = \sigma^2\mathbf{I}. \tag{1.2}$$

In other words it is a zero-mean random vector whose elements are uncorrelated and have the same standard deviation σ, so it refers to the way it is distributed, while Gaussianity refers to the probability of the signal falling within any particular range of amplitudes. The choice for this type of noise is very specific, but searching in the literature on image denoising, reveals that this noise model is very popular and covered by most of the developed algorithms. There are many explanation for this choice: for example, in imaging, noise arises due for physical reasons and usually follows a Gaussian distribution; moreover it can approximate the Poisson distribution noise, and finally, mathematically, it is simple to formulate and it is related with Minimum Mean squared error, as will be seen.

The denoising task is that of estimating $\mathbf{X}$ from $\mathbf{Y}$, knowing σ, thus a denoiser is a function of the form $\hat{\mathbf{X}} = \mathcal{D}(\mathbf{Y}, \sigma)$. To recover the nearest estimate of $\mathbf{X}$, the denoiser has to be trained in a wise way, to achieve the best performance in terms of computational cost, error and quality, in order to obtain a clean image without losing important details. The most commonly used metric is the Mean Squared Error (MSE), which computes the expected error over the image distribution:

$$MSE = \mathbb{E}(\|\mathbf{X} - \hat{\mathbf{X}}\|_2^2) = \mathbb{E}(\|\mathbf{X} - D(\mathbf{Y}, \sigma)\|_2^2). \tag{1.3}$$

However, the choice of the error metric is of crucial importance and will be discussed later in detail. As anticipated, the goal of the denoiser is to attenuate the noise, but this process can degrade the image content by losing some details, such as edges, or damage the image texture, resulting in a blurry version. In the next sections, we will describe one of the algorithms used for denoising, which is formulated starting from a classical algorithm, the K-SVD and then improved to a learnable version. Its strengths and weaknesses will be discussed.

1.2 K-SVD denoiser

Task: Denoise a given image Y form white and additive Gaussian white noise with standard deviation σ.

Algorithm Parameters: n - block size, k - dictionary size, J - number of training iterations, λ - Lagrange multiplier, and C - noise gain.

$$\min_{\mathbf{X},\mathbf{D},\mathbf{A}} \left\{ \lambda \|\mathbf{Y}-\mathbf{X}\| + \sum_{ij} \mu_{ij} \|\alpha_{ij}\|_0 + \sum_{ij} \|\mathbf{D}\alpha_{ij} - R_{ij}X\|_2^2 \right\}$$

1. Initialization : Set $\mathbf{X} = \mathbf{Y}$, $\mathbf{D}$ = overcomplete DCT dictionary.
2. Repeat J times:
   - *Sparse Coding Stage*: Use any pursuit algorithm to compute the representation vectors α_{ij} for each patch $R_{ij}\mathbf{X}$, by approximating the solution of

     $$\forall_{ij} \ \min_{\alpha_{ij}} \|\alpha_{ij}\|_0 \quad \text{s.t.} \quad \|Rij\mathbf{X} - \mathbf{D}\alpha_{ij}\|_2^2 \leq (C\sigma)^2.$$

   - *Dictionary Update Stage*: For each column $l = 1, 2, \ldots, k$ in $\mathbf{D}$, update it by
     - Find the set of patches that use this atom, $\omega_l = \{(i,j) | \alpha_{ij}(l) \neq 0\}$.
     - For each index $(i,j) \in \omega_l$, compute its representation error

       $$\mathbf{e}_{ij}^l = R_{ij}\mathbf{X}_{ij} - \sum_{m \neq l} \mathbf{d}_m \alpha_{ij}(m).$$

     - set $\mathbf{E}_l$ as the matrix whose columns are $\{\mathbf{e}_{ij}^l\}_{(i,j)\in\omega_l}$
     - Apply SVD decomposition $\mathbf{E}_l = \mathbf{U}\mathbf{\Delta}\mathbf{V}^T$. Choose the updated dictionary column $\tilde{\mathbf{d}}_l$ to be the first column of $\mathbf{U}$. Update the coefficient values $\{\alpha_{ij}(l)\}_{(i,j)\in\omega_l}$ to be the entries of $\mathbf{V}$ multiplied by $\mathbf{\Delta}(1,1)$.
3. Set:

   $$X = \left(\lambda\mathbf{I} + \sum_{ij} R_{ij}^T R_{ij}\right)^{-1} \left(\lambda\mathbf{Y} + \sum_{ij} R_{ij}^T \mathbf{D}\alpha_{ij}\right)$$

Figure 1.1: structure of the classical K-SVD algorithm for denoising (2)

In this section, we will briefly describe the K-SVD denoising algorithm, as described in (2) in 2006, state of-the-art classical algorithm for many years, it was later improved in its deep version (1) in 2021, to be compared to the most recent learnable denoising networks.

The model works with images $\mathbf{X}$ to which white Gaussian noise σ is added, obtaining $\mathbf{Y}$ as shown in equation 1.1. Therefore, the input of the algorithm (shown in figure 1.1) will be the corrupted image $\mathbf{Y}$, and the image to be obtained is set as $\mathbf{X} = \mathbf{Y}$, which is then decomposed into small patches such that $\mathbf{x}_k = \mathbf{R}_k\mathbf{X}$. For each of these patches, the algorithm obtains their sparse representation α_k using an overcomplete dictionary $\mathbf{D}$, which is updated using SVD decomposition. Finally, the algorithm restores the image $\mathbf{X}$ using this updated dictionary.

In the formulation of the K-SVD algorithm for denoising (2), the global Maximum a Posteriori (MAP) estimator for the denoising is given as a minimizer of a well-defined global penalty term which takes into account sparseness and proximity. It is defined by:

$$\min_{\{\alpha_k\}_k,\mathbf{X}} \frac{\mu}{2}\|\mathbf{X}-\mathbf{Y}\|_2^2 + \sum_k (\lambda_k\|\alpha_k\|_0 + \frac{1}{2}\|\mathbf{D}\alpha_k - \mathbf{R}_k\mathbf{X}\|_2^2). \tag{1.4}$$

As we can see from the first term, there needs to have proximity between the measured image $\mathbf{Y}$ and its denoised version $\mathbf{X}$, while the second term controls that in the constructed image $\mathbf{X}$, every patch $\mathbf{x}_k$ has a sparse representation, which will be assured in the structure of the network. Assuming then that the dictionary $\mathbf{D}$ is known, in the objective function 1.4 we have to know the sparse representation α_k for each patch k, and the measured image $\mathbf{X}$. Given that $\mathbf{X} = \mathbf{Y}$, the minimization algorithm seeks to ensure the optimal sparse representation for every patch k, controlled by the error $\|\mathbf{D}\alpha_k - \mathbf{R}_k\mathbf{X}\|_2^2$. The minimization problem is then:

$$\hat{\alpha}_k = \arg\min_{\alpha} \|\alpha_k\|_0 \quad s.t. \quad \|\mathbf{D}\alpha_k - \mathbf{y}_k\|_2^2 \leq p\sigma^2, \tag{1.5}$$

which is solved in (2) using Ortonormal Matching Pursuit (OMP) when the second term goes below $p\sigma^2$. Once the sparse representation $\{\hat{\alpha}_k\}_k$ is obtained, the second part of 1.4 has to be solved

$$\hat{\mathbf{X}} = \arg\min_{\mathbf{X}} \frac{\mu}{2}\|\mathbf{X}-\mathbf{Y}\|_2^2 + \sum_k \frac{1}{2}\|\mathbf{D}\hat{\alpha}_k - \mathbf{R}_k\mathbf{X}\|_2^2, \tag{1.6}$$

whose solution is close to the form:

$$\hat{\mathbf{X}} = \left(\sum_k \mathbf{R}_k^T\mathbf{R}_k + \mu I\right)^{-1}\left(\mu Y + \sum_k \mathbf{R}_k^T\mathbf{D}\hat{\alpha}_k\right) \tag{1.7}$$

which returns the denoised image.

1.3 Introduction to Deep K-SVD denoiser

Before describing the deep version of the previous mentioned K-SVD algorithm for denoising, proposed in 2021 in (1), we will delve into the mathematical theory used for it. From equation 1.1, we obtain a noisy measurement of $\mathbf{X}$. In general terms, the original image can be written as a sparse linear combination of elementary images; thus, what we obtain is a sparse linear combination of these elementary images that has been contaminated with additive noise. Mathematically, considering a patch $\mathbf{x}_k \in \mathbb{R}^p$ of the original image and a dictionary $\mathbf{D} \in \mathbb{R}^{p \times m}$, it can be expressed as :

$$\mathbf{x}_k = \mathbf{D}\alpha_k, \tag{1.8}$$

where $\alpha_k \in \mathbb{R}^m$ is a representation of $\mathbf{x}_k$ that we want to be sparse. However, what we get is the noisy image patch $\mathbf{y}_k$ such that:

$$\mathbf{y}_k = \mathbf{D}\alpha_k + \mathbf{v}. \tag{1.9}$$

The goal of denoising using sparse representation is to find an approximation of α_k given $\mathbf{y}_k$. In synthesis we want to find a vector as sparse as possible with noise allowance. In mathematical terms

$$\hat{\alpha}_k = \arg\min_{\alpha_k} \lambda_k \|\alpha_k\|_0 + \frac{1}{2}\|\mathbf{D}\alpha_k - \mathbf{R}_k\mathbf{X}\|_2^2 \tag{1.10}$$

which is equivalent to 1.5. Unfortunately the L_0 formulation rarely has a unique solution, as many feasible variants of it sharing the same support can be built. Therefore, an approximation algorithm has to be used; in the K-SVD algorithm, the optimal approximated representation is achieved using OMP and using the level of noise σ as a stopping criterion. In the deep version it is substituted with a learnable alternative. Another way to approximate is to find the closest convexification replacing the L_0 norm with L_1 as it is shown in (10) and (11), which can be cast as a linear programming problem.

The sparse coding 1.10 is then approximated by solving the minimization problem:

$$\hat{\alpha}_k = \arg\min_{\alpha_k} \lambda_k \|\alpha_k\|_1 + \frac{1}{2}\|\mathbf{D}\alpha_k - \mathbf{R}_k\mathbf{Y}\|_2^2 \tag{1.11}$$

where the parameter λ_k manages a trade-off between the approximation error and the sparsity of the coefficient vector. Once this formulation has been minimized to find $\hat{\alpha}_k$, it approximates the ideal coefficient vector α_k. In the architecture under consideration, 1.11 is solved using the learnable version of the Iterative Soft Thresholding Algorithm (ISTA) (12). This is a regularization method that minimizes a functional obtained by adding a penalization term to the discrepancy between the input image and the ideal one. (12) gives a generalization for the sparse coding: if we write the images as functions in a Hilbert space we have that the ideal representation 1.8 is given by

$$Kf = h \tag{1.12}$$

where K is a bounded operator from $\mathcal{H}$ to $\mathcal{H}'$, $f \in \mathcal{H}$ what we want then to be a sparse coefficient vector and $h \in \mathcal{H}'$ the image seen as a function. The noisy image is now defined as $g \in \mathcal{H}'$, satisfying:

$$g = h + e = Kf + e \tag{1.13}$$

where e represents the noise. Furthermore, the functional that the ISTA algorithm has to minimize, takes into account a penalization term, which is a weighted L^p norm of the coefficients of f, with respect to a given orthonormal basis in $\mathcal{H}$, with $1 \leq p \leq 2$ and $f \in \mathcal{H}$. Moreover, given an orthonormal basis $(\varphi_\gamma)_\gamma$ of $\mathcal{H}$, and given a sequence of strictly positive weights $\mathbf{w} = (w_\gamma)_\gamma$, the functional $\Phi_{w,p}(f)$ is defined as

$$\Phi_{w,p}(f) = \Delta(f) + \sum_\gamma w_\gamma |\langle f, \varphi_\gamma \rangle|^p = \|Kf - g\|^2 + \sum_\gamma w_\gamma |\langle f, \varphi_\gamma \rangle|^p, \tag{1.14}$$

For $p = 1$, keeping the weights fixed at μ, meaning $w_\gamma = \mu \quad \forall \gamma$, the minimization procedure promotes sparsity of the expansion of f with respect to the φ_γ, which is what we are considering. With $p = 1$ and $\langle f, \varphi_\gamma \rangle$, $\langle g, \varphi_\gamma \rangle$ written as f_γ, g_γ, in the ideal case which we want to approach, the minimizer of the functional is given by

$$f^* = \sum_\gamma f^*_\gamma \varphi_\gamma = \sum_\gamma S_\mu(g_\gamma)\varphi_\gamma, \tag{1.15}$$

where S_μ is a thresholding operator, from $\mathbb{R}$ to $\mathbb{R}$, defined as:

$$S_\mu(x) = \begin{cases} x - \dfrac{\mu}{2} \; if \; x \geq \dfrac{\mu}{2} \\ 0 \; if \; |x| < \dfrac{\mu}{2} \\ x + \dfrac{\mu}{2} \; if \; x \leq -\dfrac{\mu}{2} \end{cases} \tag{1.16}$$

The cited article (12), adopts a surrogate function instead of the defined functional 1.14. This surrogate function is subjected to minimization, and an iterative process aims to approach the minimizer. The subsequent section elaborates on these procedures, offering a generalized formulation:

Teorema 1.3.1. *Suppose the operator K maps an Hilbert space $\mathcal{H}$ to another Hilbert space $\mathcal{H}'$, with $\|K^*K\| < 1$ and suppose g is an element of $\mathcal{H}'$. Let $(\varphi_\gamma)_\gamma$ be an orthonormal basis for $\mathcal{H}$, and let $\boldsymbol{w} = (w_\gamma)_\gamma$ be a sequence of strictly positive numbers. Pick arbitrary $p \geq 1$ and $a \in \mathcal{H}$. Define the functional $\Phi^{SUR}_{w,p}(f; a)$ on $\mathcal{H}$ by*

$$\Phi^{SUR}_{w,p}(f; a) = \|Kf - g\|^2 + \sum_\gamma w_\gamma |f_\gamma|^p + \|f - a\|^2 - \|K(f - a)\|^2. \tag{1.17}$$

Then $\Phi^{SUR}_{w,p}(f; a)$ has a unique minimizer in $\mathcal{H}$. This minimizer is given by $f = S_{w,p}(a + K^*(g - Ka))$, where the operators $S_{w,p}$ are defined by

$$S_{w,p} = \sum_\gamma S_{w_\gamma, p}\varphi_\gamma, \tag{1.18}$$

with the functions $S_{w_\gamma,p}$ from $\mathbb{R}$ to itself is the thresholding defined by $(F_{w,p})^{-1}$ for $p > 1$ where $(F_{w,p}) = x + \frac{wp}{2} sign(x)|x|^{(p-1)}$ and as

$$S_{w_\gamma,1} = \begin{cases} x - \dfrac{w}{2} \ if \ x \geq \dfrac{w}{2} \\ 0 \ if \ |x| < \dfrac{w}{2} \\ x + \dfrac{w}{2} \ if \ x \leq -\dfrac{w}{2} \end{cases} \tag{1.19}$$

for $p = 1$. For all $h \in \mathcal{H}$ we have

$$\Phi^{SUR}_{w,p}(f + h; a) \geq \Phi^{SUR}_{w,p}(f; a) + \|h\|^2. \tag{1.20}$$

With the same hypothesis of the theorem, having found the minimizer of $\Phi^{SUR}_{w,p}(f; a)$, it follows

Corollary 1.3.2. *Pick $f^0 \in \mathcal{H}$, and define the functions f^n recursively by the algorithm*

$$f^0 \ arbitrary; \ f^n = \arg\min(\Phi^{SUR}_{w,p}(f; f^{n-1})) \quad n = 1, 2 \ldots.$$

Then

$$f^n = S_{w,p}(f^{n-1} - K^*(g - Kf^{n-1})). \tag{1.21}$$

For a generalization, the operator K^*K can be upper bounded by a general operator D diagonal in the φ_γ basis: $D\varphi_\gamma = d_\gamma \varphi_\gamma$; such that

$$K^*K + \eta I \leq D \ \text{ for some } \ \eta > 0,$$

and the construction 1.21 become

$$f^n = S_{w/D,p}(f^{n-1} - D^{-1}\left[K^*(g - Kf^{n-1})\right]). \tag{1.22}$$

From (12) is stated the following

Teorema 1.3.3. *Let K be a bounded operator from $\mathcal{H}$ to $\mathcal{H}$, with norm strictly bounded by 1. Take $p \in [1, 2]$, and let $S_{w,p}$be the shrinkage operator defined in 1.18, where the sequence $\boldsymbol{w} = (w_\gamma)_\gamma$ is uniformly bounded below away from zero. Then the sequence of iterates defined in 1.21, with f^0 arbitrarily chosen in $\mathcal{H}$, converges strongly to a minimizer of the functional*

$$\Phi_{\boldsymbol{w},p}(f) = \|Kf - g\|^2 + |||f|||^p_{\boldsymbol{w},p}, \tag{1.23}$$

where $|||f|||_{\boldsymbol{w},p}$ denotes the norm

$$|||f|||_{\boldsymbol{w},p} = \left[\sum_\gamma w_\gamma |\langle f, \varphi_\gamma \rangle|^p\right]^{1/p} \quad 1 \leq p \leq 2.$$

So far, we have discussed the construction of the minimizer. However, it is also important to consider the balance between the discrepancy $\|Kf - g\|^2$ and the penalty term $|||f|||^p_{\mathbf{w},p}$ in 1.23. This term, can be regularized with a term μ to which the functional depends.

$$\Phi_{\mathbf{w},p}(f) = \|Kf - g\|^2 + \mu|||f|||^p_{\mathbf{w},p}. \tag{1.24}$$

Considering this formulation, we can apply the above discussion to the problem we are taking in account. In (13) the ISTA algorithm is applied to find the optimizer of the sparse code 1.11. In our considered problem, we have to find the minimization 1.11 of the sparse code, which can be written as a functional 1.24 of the form:

$$\lambda_k\|\alpha_k\|_1 + \frac{1}{2}\|\mathbf{D}\alpha_k - \mathbf{R}_k\mathbf{Y}\|_2^2, \tag{1.25}$$

with $p = 1$.
Summarizing the preceding discussions in simpler terms, given an input vector $\mathbf{y}_k$, we can determine the minimizer of 1.25 through the following recursive iterations:

$$\hat{\alpha}^k_{t+1} = S_{\frac{\lambda_k}{c}}(\hat{\alpha}^k_t - \frac{1}{c}\mathbf{D}^T(\mathbf{D}\hat{\alpha}^k_t - \mathbf{y}_k)); \qquad \hat{\alpha}^k_0 = 0. \tag{1.26}$$

Here, the function $S_\theta(V)$ represents the component-wise function with a vector of threshold θ, as defined in 1.19. Specifically, $[S_\theta(V)]_i$ is computed as $sign(V_i)(|V_i| - \theta_i)_+$, where all thresholds are set as $\theta_i = \frac{\lambda_k}{c}$. Here, λ_k is a

Algorithm 1 ISTA

function ISTA(X, Z, W_d, α, L)
Require: $L >$ largest eigenvalue of $W_d^T W_d$.
Initialize: $Z = 0$,
repeat
$Z = h_{(\alpha/L)}(Z - \frac{1}{L}W_d^T(W_d Z - X))$
until change in Z below a threshold
end function

Figure 1.2: the structure of ISTA algorithm. Here $\mathbf{X}$ is the input image, Z the sparse vector, $\mathbf{W}_d$ the dictionary and α a parameter which control sparseness.

coefficient which regularize 1.25 and controls sparseness, while the constant c defines the upper bound on the largest eigenvalue of $\mathbf{D}^T\mathbf{D}$; in our case c will be represented by the squared spectral norm of the dictionary and the role of sparseness coefficient is taken by λ_k, which has to be chosen in an appropriate way in the algorithm. In fact this will be done in a learnable way using Multi-Layer Perceptron (MLP). Once the sparse representation for each patch has been obtained, the algorithm proceeds to reconstruct the image. However, the reconstruction method differs from that described in 1.7. Instead, it involves a weighted average of the patches.
In the next chapter the architecture of the algorithm will be explained in detail.

Chapter 2

Algorithmic Approaches

2.1 Deep K-SVD denoising

In this section, we will delve into a comprehensive examination of the deep K-SVD denoising algorithm as outlined in the preceding chapter. This detailed understanding will serve as the foundation for initiating our subsequent work. The original formulation proposed (1), focuses on removing noise from images. To better understand the algorithm operation, we will analyze its architecture, depicted in Figure 2.1, as well as examine its implementation code on a specific dataset of images. This analysis will provide insights into the inner workings of the algorithm.

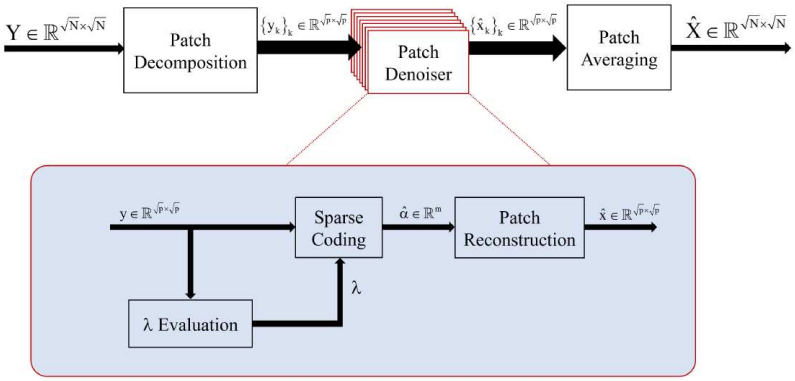

Figure 2.1

The model works with images described as matrices $\mathbf{X} \in \mathbb{R}^{\sqrt{N}\times\sqrt{N}}$, to which is added white Gaussian noise σ, obtaining $\mathbf{Y} \in \mathbb{R}^{\sqrt{N}\times\sqrt{N}}$ as in 1.1. Therefore, the input of the algorithm will be a corrupted image $\mathbf{Y}$, which is then decomposed into small overlapping patches of size $\sqrt{p} \times \sqrt{p}$ pixels, ordered as vectors

$\mathbf{y}_k \in \mathbb{R}^p$ such that $\mathbf{y}_k = \mathbf{R}_k\mathbf{Y}$, with $\mathbf{R}_k \in \mathbb{R}^{p\times N}$ an operator to select the patches and $\mathbf{Y}$ reshaped to be a vector of dimension N. For each of these patches the algorithm obtains their sparse representation α_k using a learned dictionary $\mathbf{D}$ and averaging these overlapping patches, the algorithm restore the image $\hat{\mathbf{X}} \in \mathbb{R}^{\sqrt{N}\times\sqrt{N}}$.

2.2 Architecture

The core of the model (figure 2.1) is the patch denoiser which will be described below in three stages: sparse coding, λ evaluation and patch reconstruction.

2.2.1 Sparse coding

Given all patches $\mathbf{y}_k$ of a corrupted image $\mathbf{Y}$, during the sparse coding stage the algorithm must compute a sparse vector for each of these vectors patches. This is done as we have described in the previous chapter, iterating 1.26 until $t+1 = T$ for a given value of T. As we have shown, this approximates the sparse representation of the related clean patch $\mathbf{x}_k$, trying to minimize the noise. This is done accordingly to a learned dictionary $\mathbf{D} \in \mathbb{R}^{p\times m}$, as it is done in the original K-SVD algorithm. However, in the deep version we obtain the sparse coding in a learnable way. In simpler terms, after T iterations, we have obtained the following using the recursive equation 1.26:

$$\hat{\alpha}_T = S_{\frac{\lambda}{c}}\left(\hat{\alpha}_{T-1} - \frac{1}{c}\mathbf{D}^T(\mathbf{D}\hat{\alpha}_{T-1} - \mathbf{y})\right).$$

In this context, we treat $\mathbf{y}$ as the collection of all patches $\mathbf{y}_k$, and similarly, $\hat{\alpha}_t$ and λ are considered for every k. Additionally, c represents the squared spectral norm of $\mathbf{D}$, as it serves as the upper bound of the eigenvectors of $\mathbf{D}^T\mathbf{D}$. This treatment is applied to each patch individually during the execution of the algorithm. During this phase the network tries to find an estimate of every patch $\mathbf{x}_k$ of $\mathbf{X}$, given the noisy patch $\mathbf{y}_k$, minimizing the discrepancy term $\|\mathbf{D}\alpha_k - \mathbf{R}_k\mathbf{Y}\|_2^2$. What differs respect to the previously described ISTA algorithm, is the choice of the regularization parameter λ, which in order to quantify better the tradeoff between sparseness and discrepancy, has to be learned. This is done using MLP and will be discussed in the next paragraph. Moreover, in order to make the ISTA learnable, the code fixes the number of iterations T and treats the dictionary $\mathbf{D}$ and its squared spectral norm c as learnable parameters. These parameters are updated during the backpropagation process of the algorithm.

2.2.2 λ Evaluation

To optimize sparse coding, the model needs to determine the appropriate regularization parameter λ. This is achieved by training a Multi-Layer Perceptron (MLP) network from the patches $\mathbf{y}_k$ to their corresponding regularization parameters λ_k for each k: $\lambda = f_\theta(\mathbf{y})$.

The MLP network consists of an input layer with p nodes, followed by three hidden layers. Each hidden layer comprises a fully connected linear mapping followed by a Rectified Linear Unit (ReLU) activation function, defined as:

$$\text{ReLU}(x) = \begin{cases} x \ if \ x \geq 0 \\ 0 \ otherwise \end{cases}$$

Finally, the output layer consists of a single node representing λ. The structure is the following:

$$\mathbf{y} \rightarrow [2p \times p] \rightarrow \text{ReLU} \rightarrow [p \times 2p] \rightarrow \text{ReLU} \rightarrow [p/2 \times p] \rightarrow \text{ReLU} \rightarrow [1 \times p] \rightarrow \text{Act} \rightarrow \lambda \tag{2.1}$$

where $[a \times b]$ symbolizes a multiplication by a matrix of that size. Going more in depth we have

$$\nu = \text{ReLU}(\begin{bmatrix} b_0 & A_0 \end{bmatrix} \begin{bmatrix} 1 \\ y \end{bmatrix}) \tag{2.2}$$

$$\mu = \text{ReLU}(\begin{bmatrix} b_1 & A_1 \end{bmatrix} \begin{bmatrix} 1 \\ \nu \end{bmatrix}) \tag{2.3}$$

$$\eta = \text{ReLU}(\begin{bmatrix} b_2 & A_2 \end{bmatrix} \begin{bmatrix} 1 \\ \mu \end{bmatrix}) \tag{2.4}$$

$$\lambda = \text{Act}(\begin{bmatrix} b_3 & A_3 \end{bmatrix} \begin{bmatrix} 1 \\ \eta \end{bmatrix}). \tag{2.5}$$

In the formulation given in (1) the output layer λ, is not followed by a ReLU, while in our formulation we will put the option to add the proper activation function, that here is denoted as Act(); so to be more precise in the original formulation Act() is the identity function. However, as the input layer can achieve negative values, they can be propagated in the output layer. For that reason, to avoid negative values for λ and make the sparse code operate correctly, in the simplest way we can apply a ReLU to it, moving all the values in the positive range.

2.2.3 Patch reconstruction

Once the appropriate λ is obtained, the sparse representation $\hat{\alpha}$ of the patches can be computed using the dictionary $\mathbf{D}$ within a fixed number T of iterations of the ISTA algorithm. Consequently, the network reconstructs the cleaned version $\hat{\mathbf{x}}$ of $\mathbf{y}$ by calculating $\hat{\mathbf{x}} = \mathbf{D}\hat{\alpha}$, which approximates 1.8.

2.2.4 Patch Averaging

Summing it all together, the network begins by dividing the corrupted image $\mathbf{Y}$ into fully overlapping patches using the operators $\{\mathbf{R}_k\}_k$. Then, in the sparse stage, each of these patches is denoised, resulting in $\hat{\mathbf{x}}$.
The final stage consists in averaging all these cleaned patches together, using a

learned weighted combination of them. The reconstructed image is obtained in the following way:

$$\hat{\mathbf{X}} = \frac{\sum_k \mathbf{R}_k^T(\hat{\mathbf{x}} \circ \mathbf{w})}{\sum_k \mathbf{R}_k^T \mathbf{w}}, \tag{2.6}$$

where $\mathbf{w} \in \mathbb{R}^p$ weights the values of the reconstructed patch, and is one of the learnable parameters of the net.

2.3 Learnable Parameters Initialization

The initialization of learnable parameters is a crucial step to optimize the performance of the network and prevent issues such as exploding or vanishing weights during backpropagation. In our model the weights to be learned are the dictionary $\mathbf{D}$, its squared spectral norm c, the averaging weight $\mathbf{w}$ for the image reconstruction and finally the matrices $\mathbf{A}_i$ with biases $\mathbf{b}_i$, for $i = 0, 1, 2, 3$ in the MLP. They will be initialized as follows.

2.3.1 Dictionary settings

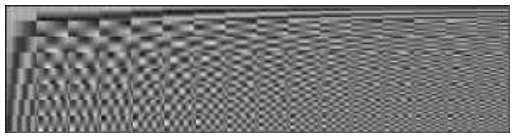

Figure 2.2: Overcomplete Discrete Cosinus Transform of size 64×256

In (1) and (2) the dictionary is initialized as the 2-D Overcomplete Discrete Cosinus Transform (ODCT), which is a variant of the Discrete Cosinus Transform (DCT) that uses an overcomplete set of basis functions. In our implementation of the algorithm we built the 2-D Discrete Cosinus Transform (DCT) consisting in taking the 1-D DCT performed along the rows (1 dimension) and then along the columns:

$$X_{ij} = \sum_{k=0}^{m-1} \left(\sum_{l=0}^{m-1} x_{kl} \cos\left(\frac{\pi}{m} lj\right) \right) \cos\left(\frac{\pi}{m} ki\right). \tag{2.7}$$

To that purpose in the implementation we use the Kronecker product of the 1-D basis functions against themselves, to combine each basis function with every other basis function. This assure to obtain a new set of basis functions extending them into two dimensions (row and columns). Indeed, this initialization process is crucial for achieving redundancy, as it enables the resulting dictionary to contain more basis functions than the original set. This ensures the construction of an overcomplete dictionary, which is essential for capturing all the patterns present in the data, especially in the context of synthetic image generation. An illustrative example showcasing the construction of the dictionary is provided

in figure 2.2. From the dictionary we also obtain the parameter c, which has to bound the largest eigenvalue of $\mathbf{D}^T\mathbf{D}$ in the LISTA algorithm; to that purpose it is chosen to be the squared spectral norm of $\mathbf{D}$.

2.3.2 Averaging weight settings

In line with the original implementation methodology, we have employed random initialization for the averaging weight $\mathbf{w}$ utilized in image reconstruction. Specifically, $\mathbf{w}$ is initialized following a normal distribution $\mathbf{w} \sim \mathcal{N}(1, \frac{1}{10}\mathbf{1}_p)$, where $\mathbf{1}_p$ denotes a vector of ones of dimension p. This initialization strategy has been chosen for its efficacy in handling both the forward and backward processes of training, thus mitigating potential issues such as vanishing or exploding gradients. Furthermore it helps maintaining diversity among the weight, offering efficacy in the learning process.

2.3.3 MLP initialization

To initialize the Multi-Layer Perceptron (MLP) for λ evaluation, the matrices $\mathbf{A}_i$ and biases $\mathbf{b}_i$ are randomly initialized using a uniform distribution $\mathcal{U}(-\sqrt{n}, \sqrt{n})$, where n denotes the size of the incoming node. Specifically, if the incoming node is represented as $x \in \mathbb{R}^n$ and the outgoing node as $y \in \mathbb{R}^m$, the matrix operation is defined as:

$$y = \begin{bmatrix} b & A \end{bmatrix} \begin{bmatrix} 1 \\ x \end{bmatrix}$$

where $A \in \mathbb{R}^{m \times n}$ represents the matrix and $b \in \mathbb{R}^m$ represents the bias vector. Both A and b are computed with entries drawn randomly from the uniform distribution.

2.4 Training

So far, we have illustrated how the model operates: given a corrupted image $\mathbf{Y}$ as input, it returns its denoised version $\hat{\mathbf{X}}$. Referring to the denoising algorithm as $\mathcal{D}_\Theta$, where Θ represents the learnable parameters of the net, we express this relationship as $\hat{\mathbf{X}} = \mathcal{D}_\Theta(\mathbf{Y}, \sigma)$.
The final step involves training the model, minimizing a loss function $\varepsilon_\Theta = \sum_{k=1}^{N} dist(x_k, \hat{x}_k) = \sum_{k=1}^{N} dist(x_k, \mathcal{D}_\Theta(y_k, \sigma))$. Here, $dist(x, \hat{x})$ represents a pixel-wise distance function between the clean image $\mathbf{X}$ and the denoised image $\hat{\mathbf{X}}$, aiming to obtain a denoised image as close as possible to the original $\mathbf{X}$.
This is accomplished through backpropagation, where gradients of ε with respect to each of the learnable weights of the network are computed and updated using an appropriate optimizer.
For the training process, the dataset and certain parameters need to be set. In the article (1) and the corresponding code (3), the Berkeley Segmentation Dataset (BSDS) is utilized. This dataset comprises 432 images for training and

68 for testing, all converted to grayscale, with the two sets being completely disjoint.

2.4.1 Training Dataset

Figure 2.3: Original clean image and randomly selected cropped image

Figure 2.4: Process of denoising cropped images during training; from left to right: clean subimage, noisy subimage and denoised subimage.

From the training set of images, which are of dimensions $W \times H$ and may be vertically or horizontally oriented, we extract cropped images to a size of $\sqrt{N} \times \sqrt{N}$ pixels (see figure 2.3). Gaussian noise with zero mean and a specified level of noise σ is then added to each of these cropped images. The subimages are obtained from a list of all training images. By specifying an index, a specific image is selected, and given the dimensions of the desired subimage, it is then extracted. This process yields a dataset composed of pairs of clean subimages and their corresponding noisy versions, transformed to be within the range of values $[-1, 1]$ to deal better with values. During the training process, the dataset is loaded randomly, with the option to specify the batch size. One cropped image at a time is selected, and as described above, a random pair of clean/noised subimages is chosen, and the noised one is taken as input for the network.
For the loss function, we adopt at first the Mean Squared Error (MSE) function, defined as $\varepsilon^{MSE} = \frac{1}{N}\sum_n (x_n - \hat{x}_n)^2$, where x_n represents the clean image pixel and $\hat{x}_n$ represents the denoised image pixel. This loss function is mini-

mized through backpropagation using the ADAM Stochastic Gradient Descent optimizer, with the learning rate set to $lr = 10^{-4}$, $\beta_1 = 0.9$, $\beta_2 = 0.999$, and $\epsilon = 10^{-16}$. We update the weights Θ of the network after each mini-batch, whose dimension is one cropped image.
The choice of the training function is crucial to achieving optimal performance in noise removal without sacrificing important image information. Therefore, in the next paragraph, we will discuss this in detail and propose alternative approaches.

2.4.2 Forward propagation

Before discussing backpropagation, let's summarize what the network does during the forward pass, as outlined in Section 2.2. Given a noised subimage (or subimages, depending on the batch size) $\mathbf{Y}$ as input, it operates the following process:

1. Patch Decomposition: the input image $\mathbf{Y}$ is decomposed into small patches of dimension $\sqrt{p}\times\sqrt{p}$. This is achieved using a specific operator $\mathbf{R}_k$, which iterates through the rows and columns of the $\sqrt{N} \times \sqrt{N}$ input image, selecting all possible patches.

2. Sparse Coding: each patch y_k obtained from the input is processed through MLP to obtain its corresponding regularization parameter λ_k. These parameters are then used in the ISTA to obtain the cleaned patches.

3. Denoised Image Reconstruction: the denoised patches are averaged using equation 2.6 to obtain the denoised image (see Figure 2.4). This denoised image serves as the output of the forward pass and will be utilized in the subsequent backward step.

2.4.3 Backward Propagation

During backpropagation, we utilize an optimizer to minimize the error function and update the weights. We process one mini-batch at a time, with each mini-batch containing one cropped image. To apply the optimizer, we need to compute the gradients of the loss function ε with respect to the learning parameters of the model for every patch k, for every hidden layer t, and for the last one T. This involves computing $\bigtriangledown_{\Theta^{(t)}}\varepsilon$, where $\Theta^{(t)}$ represents the parameters of the model at layer t. Additionally, we compute the gradient for the sparse code: $\bigtriangledown_{a^k_{t+1}}\varepsilon$. At layer T, we compute the gradients as follows:

- **Gradient with respect to the sparse representation a^k_T:**

$$\bigtriangledown_{a^k_T}\varepsilon = 2\,\mathbf{D}^{(T)^T}\left(\mathbf{R}_k^T \cdot \frac{\hat{\mathbf{X}} - \mathbf{X}}{\sum_k \mathbf{R}_k^T \mathbf{w}} \circ \mathbf{w}\right),$$

- **Gradient with respect to the dictionary $\mathbf{D}^{(T)}$**:

$$\bigtriangledown_{\mathbf{D}^{(T)}}\varepsilon = 2\sum_k \left(\mathbf{R}_k \cdot \frac{\hat{\mathbf{X}}-\mathbf{X}}{\sum_k \mathbf{R}_k^T\mathbf{w}} \circ \mathbf{w}\right) {a_T^k}^T,$$

- **Gradient with respect to the weight $\mathbf{w}$**:

$$\bigtriangledown_{\mathbf{w}}\varepsilon = 2\sum_k \left[\left[\left(\mathbf{R}_k \cdot \frac{\hat{\mathbf{X}}-\mathbf{X}}{\sum_k \mathbf{R}_k^T\mathbf{w}}\right) \circ \hat{\mathbf{x}}_k\right] - \mathbf{R}_k \cdot \left[\frac{\hat{\mathbf{X}}-\mathbf{X}}{(\sum_k \mathbf{R}_k^T\mathbf{w})^2} \circ \left(\sum_k \mathbf{R}_k^T(\hat{\mathbf{x}_k} \circ \mathbf{w})\right)\right]\right].$$

To recursively compute the gradients with respect to the parameters in the hidden layers, we can employ the chain rule of calculus and propagate the gradients backwards through the layers of the network. Let's break down the recursive computation of gradients with respect to the parameters in the hidden layers.

- **Gradient with respect to the Dictionary $\mathbf{D}^{(t)}$**:

$$\bigtriangledown_{\mathbf{D}^{(t)}}\varepsilon = \left(\frac{\partial\varepsilon}{\partial(D^{(t)})_{p,m}}\right)_{p,m}$$

which, using the chain rule is

$$\frac{\partial\varepsilon}{\partial(D^{(t)})_{p,m}} = \sum_k\sum_j \frac{\partial\varepsilon}{\partial(a_{t+1}^k)_j}\frac{\partial(a_{t+1}^k)_j}{\partial(D^{(t)})_{p,m}}$$

and we will compute the first term of the sum recursively with a function $\bigtriangledown_{a_t^k}\varepsilon = F(\bigtriangledown_{a_{t+1}^k}\varepsilon)$:

$$\bigtriangledown_{a_t^k}\varepsilon = (\mathbb{I}_m - \frac{1}{c^{(t)}}{\mathbf{D}^{(t)}}^T\mathbf{D}^{(t)})(\bigtriangledown_{a_{t+1}^k}\varepsilon \circ \partial_v S) \quad \forall k,$$

while the second can be computed by hand, which bring us to the recursive function $\bigtriangledown_{\mathbf{D}^{(t)}}\varepsilon = G(\bigtriangledown_{a_{t+1}^k}\varepsilon)$:

$$\bigtriangledown_{\mathbf{D}^{(t)}}\varepsilon = -\frac{1}{c^{(t)}}\left[\sum_k (\mathbf{D}^{(t)}a_t^k - \mathbf{y}_k)(\bigtriangledown_{a_{t+1}^k}\varepsilon \circ \partial_v S)^T + \sum_k \mathbf{D}^{(t)}(\bigtriangledown_{a_{t+1}^k}\varepsilon \circ \partial_v S)\,(a_t^k)^T\right]$$

with $v = \left[a_t^k - \frac{1}{c^{(t)}}{\mathbf{D}^{(t)}}^T(\mathbf{D}^{(t)}{a_t}^k - \mathbf{y}_k)\right]$ the second argument of the soft thresholding 1.26 written as $S(\theta, v)$.

- **Gradient with respect to the parameter $c^{(t)}$**:
Similarly, we compute the gradient respect to the parameter $c^{(t)}$ at layer t:

$$\bigtriangledown_{c^{(t)}}\varepsilon = \frac{\partial\varepsilon}{\partial(c^{(t)})} = \sum_k\sum_j \frac{\partial\varepsilon}{\partial(a_{t+1}^k)_j}\frac{\partial(a_{t+1}^k)_j}{\partial c^{(t)}}$$

finding

$$\nabla_{c^{(t)}}\varepsilon = \frac{1}{{c^{(t)}}^2}\sum_k \left((\nabla_{a_{t+1}^k}\varepsilon \circ \partial_v S)^T \cdot ({\mathbf{D}^{(t)}}^T(\mathbf{D}^{(t)}{a_t}^k - \mathbf{y}_k)) \right) - {\lambda_k}^{(t)}\,\mathrm{ONES}(1\times m)\cdot(\nabla_{a_{t+1}^k}\varepsilon \circ \partial_\theta S).$$

- **Gradients with respect to the MLP parameters $\mathbf{b}_i\mathbf{A}_i{}^{(t)}$**: Finally, in the same way:

$$\nabla_{\mathbf{b}_i\mathbf{A}_i{}^{(t)}}\varepsilon = \left(\frac{\partial\varepsilon}{\partial(b_iA_i^{(t)})_{r,c}}\right)_{r,c},$$

where

$$\frac{\partial\varepsilon}{\partial(b_i{A_i}^{(t)})_{r,c}} = \sum_k\sum_j \frac{\partial\varepsilon}{\partial(a_{t+1}^k)_j}\frac{\partial(a_{t+1}^k)_j}{\partial\lambda_k^{(t)}}\frac{\partial\lambda_k^{(t)}}{\partial(b_i{A_i}^{(t)})_{r,c}}$$

Therefore, considering the equations 2.5, 2.4, 2.3, 2.2, we obtain

$$\nabla_{\mathbf{b}_3\mathbf{A}_3{}^{(t)}}\varepsilon = \sum_k ARG\cdot\left(ARG_\eta\cdot\begin{bmatrix}1\\ \eta^{(t)}\end{bmatrix}^T\right),$$

$$\nabla_{\mathbf{b}_2\mathbf{A}_2{}^{(t)}}\varepsilon = \sum_k ARG\cdot\left(ARG_\mu\cdot\begin{bmatrix}1\\ \mu^{(t)}\end{bmatrix}^T\right),$$

$$\nabla_{\mathbf{b}_1\mathbf{A}_1{}^{(t)}}\varepsilon = \sum_k ARG\cdot\left(ARG_\nu\cdot\begin{bmatrix}1\\ \nu^{(t)}\end{bmatrix}^T\right),$$

$$\nabla_{\mathbf{b}_0\mathbf{A}_0{}^{(t)}}\varepsilon = \sum_k ARG\cdot\left(ARG_y\cdot\begin{bmatrix}1\\ \mathbf{y}_k\end{bmatrix}^T\right),$$

where

$$ARG_\eta = Act'\left(\begin{bmatrix}b_3 & A_3\end{bmatrix}^{(t)}\begin{bmatrix}1\\ \eta^{(t)}\end{bmatrix}\right)$$

$$ARG_\mu = ReLU'\left(\begin{bmatrix}b_2 & A_2\end{bmatrix}^{(t)}\begin{bmatrix}1\\ \mu^{(t)}\end{bmatrix}\right)\circ\left({A_3^{(t)}}^T\cdot ARG_\eta\right)$$

$$ARG_\nu = ReLU'\left(\begin{bmatrix}b_1 & A_1\end{bmatrix}^{(t)}\begin{bmatrix}1\\ \nu^{(t)}\end{bmatrix}\right)\circ\left({A_2^{(t)}}^T\cdot ARG_\mu\right)$$

$$ARG_y = ReLU'\left(\begin{bmatrix}b_0 & A_0\end{bmatrix}^{(t)}\begin{bmatrix}1\\ \mathbf{y}_k\end{bmatrix}\right)\circ\left({A_1^{(t)}}^T\cdot ARG_\nu\right)$$

$$ARG = \frac{1}{c^{(t)}}\left[ONES(1\times m)\cdot\left(\nabla_{a_{t+1}^k}\varepsilon\circ\partial_\theta S\right)\right]$$

After computing gradients for each layer t and parameter within each layer, they are aggregated over all layers using the chain rule: $\bigtriangledown_{\Theta}\varepsilon = \sum_t \bigtriangledown_{\Theta^{(t)}}\varepsilon$. This combined gradient represents the overall gradient of the loss function ε with respect to all the learnable parameters Θ of the entire neural network. Subsequently, this aggregated gradient is utilized to update all parameters within the optimizer.

2.4.4 Optimizer

Having computed the gradients of the loss function respect to every weight, they can be optimized using the proper algorithm. We consider at first the Stochastic Gradient Descent (SGD), then for better results we consider the Adaptive Moment estimation (ADAM) (fig.2.5), in which the learning rate is adjusted respect to the parameter to be updated through momentum adaptation. In

---

input : γ (lr), β_1, β_2 (betas), θ_0 (params), $f(\theta)$ (objective)
λ (weight decay), *amsgrad*, *maximize*
initialize : $m_0 \leftarrow 0$ (first moment), $v_0 \leftarrow 0$ (second moment), $\widehat{v_0}^{max} \leftarrow 0$

---

for $t = 1$ **to** ... **do**
  if *maximize* :
    $g_t \leftarrow -\nabla_\theta f_t(\theta_{t-1})$
  else
    $g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$
  if $\lambda \neq 0$
    $g_t \leftarrow g_t + \lambda\theta_{t-1}$
  $m_t \leftarrow \beta_1 m_{t-1} + (1-\beta_1) g_t$
  $v_t \leftarrow \beta_2 v_{t-1} + (1-\beta_2) g_t^2$
  $\widehat{m_t} \leftarrow m_t / \left(1-\beta_1^t\right)$
  $\widehat{v_t} \leftarrow v_t / \left(1-\beta_2^t\right)$
  if *amsgrad*
    $\widehat{v_t}^{max} \leftarrow \max(\widehat{v_t}^{max}, \widehat{v_t})$
    $\theta_t \leftarrow \theta_{t-1} - \gamma\widehat{m_t} / \left(\sqrt{\widehat{v_t}^{max}} + \epsilon\right)$
  else
    $\theta_t \leftarrow \theta_{t-1} - \gamma\widehat{m_t} / \left(\sqrt{\widehat{v_t}} + \epsilon\right)$

---

return θ_t

Figure 2.5: Structure of the ADAM algorithm

fact what we want is to minimize the loss function respect to all the learnable parameters, so we want that the gradients approach zero. In SGD the weights are updated in the following way

$$\Theta = \Theta - lr \cdot \bigtriangledown_{\Theta}\varepsilon$$

with lr the learning rate appropriately chosen. In our algorithm, as in (1), we used one cropped image as mini-batch picked randomly from the dataset. In the ADAM algorithm the learning rate and the gradients are adapted to the parameter, so at every update, with batch size equal to 1, the first and second momentum, previous set to be zero for initialization, are updated at every iteration as

$$m_i = \beta_1 m_{i-1} + (1-\beta_1) \cdot \bigtriangledown_{\Theta}\varepsilon,$$

$$v_i = \beta_2 v_{i-1} + (1-\beta_2) \cdot (\nabla_{\Theta}\varepsilon)^{\circ 2};$$

where $(\nabla_{\Theta}\varepsilon)^{\circ 2}$ denotes the element-wise square of the gradient of ε, and the gradients are updated using

$$\hat{m}_i = \frac{m_i}{1-\beta_1^i},$$

$$\hat{v}_i = \frac{v_i}{1-\beta_2^i}$$

as

$$\bigtriangledown_{\Theta}\varepsilon_i = \frac{\hat{m}_i}{\sqrt{\hat{v}_i} + \epsilon}.$$

Finally the weights are substituted by

$$\Theta = \Theta - lr \cdot \bigtriangledown_{\Theta}\varepsilon_i.$$

In the end, having updated the weights correctly, using them in the forward propagation, it should return the image correctly denoised and reconstructed. However the performance of the algorithm is something that can be addressed to the correct choice of the loss function: In fact the goal is to remove all the possible noise, but also reconstruct images as close as possible to the original ones, preserving small details, textures and avoiding splotchy artifacts and blurry structures. This choice will be discussed better in the next paragraph.

2.5 Loss Function

Denoisers are frequently assessed using MSE 1.3. This metric offers several advantages, such as yielding zero when the denoiser perfectly restores the image and being easily differentiable during backpropagation. Additionally, in many libraries, like `PyTorch`, MSE is one of the few pre-built loss functions. However, depending on the simplicity of the clean image manifold—indicating the level of image complexity—MSE can sometimes produce overly blurry images that fall outside this manifold. For instance, in figure 2.6, it is evident that certain

Figure 2.6: image noised, original and denoised, $MSE < 0.004$

details, such as the mountain profile in the denoised image, lack sharpness compared to the original, and the sky's texture appears noticeably rough, despite the low MSE value. This highlights the necessity of defining a criterion for assessing denoising techniques that not only aims to minimize the error caused by noise but also ensures the preservation of intricate image structures such as edges, fine details, and textures. It emphasizes the need to create evaluation methods or loss functions that consider both reducing noise and maintaining the fine details and features of the original image. For instance, the Structural Similarity Index Measure (SSIM) (4) serves to compare the structures and patterns of two images, evaluating their similarity in luminance, contrast, and structures. Mathematically, it is expressed as:

$$SSIM(\mathbf{X}, \hat{\mathbf{X}}) = \frac{2\mu_{\mathbf{X}}\mu_{\hat{\mathbf{X}}} + C_1}{\mu_{\mathbf{X}}^2 + \mu_{\hat{\mathbf{X}}}^2 + C_1} \cdot \frac{2\sigma_{\mathbf{X}\hat{\mathbf{X}}} + C_2}{\sigma_{\mathbf{X}}^2 + \sigma_{\hat{\mathbf{X}}}^2 + C_2} = l(\mathbf{X}, \hat{\mathbf{X}}) \cdot cs(\mathbf{X}, \hat{\mathbf{X}}). \quad (2.8)$$

This formulation, interpreted as a loss function, will be discussed in more detail later. In their work (5), the authors explore the potential substitution of MSE with alternative metrics, beginning with SSIM. They then introduce additional alternatives such as Multi-Scale Structural Similarity (MS-SSIM), and discuss the possibility of combining various metrics to formulate a more suitable loss function within the domain of image processing.We aim to investigate whether adopting alternative metrics, particularly SSIM and MS-SSIM, could lead to enhanced outcomes for the implemented Deep K-SVD algorithm. Initially, we will delve into theoretical considerations to find the potential benefits of these metrics. Subsequently, we will experiment with the network to empirically validate our theoretical insights and assess the practical efficacy of incorporating these metrics into the algorithm. In the subsequent paragraph, we will delve into a more detailed illustration of the concepts previously discussed. Specifically, we will specify the significance of MSE as a loss function and discuss what constitutes a desirable value for it, often referred to as the Minimum Mean Squared Error (MMSE). Furthermore, we will compare MSE with other loss

functions to gain a comprehensive understanding of their respective strengths and limitations.

2.5.1 Minimum Mean Squared Error

As we have seen in Section 1.1, the MSE function can be stated as

$$MSE = \mathbb{E}(\|\mathbf{X} - \hat{\mathbf{X}}\|_2^2) = \frac{1}{N}\sum_i (X_i - \hat{X}_i)^2, \tag{2.9}$$

and the less the value of MSE is, the better estimator $\hat{\mathbf{X}}$ is. If the noise is an additive Gaussian noise with zero mean and variance σ^2, as in our considered case, we can find a relation between the clean/denoised image distance (the MSE) and the variance of the noise. Putting all together we want to minimize the value of MSE, and finding the estimator $\hat{\mathbf{X}}$ such that the function achieves such minimum, helping us to understand how well noise can be removed from a given image. This is independent from the denoiser we are using. So first of all we will find a way to find the best estimator $\hat{\mathbf{X}}_{MMSE}$ and the related MSE value, to set a parameter to understand if our model is able to achieve the best performance in terms of noise removal which, as discussed before, does not guarantee structural perfection of the image. We want, in fact, to find

$$\hat{\mathbf{x}}_{MMSE} = \arg\min_{\hat{\mathbf{x}}} \mathbb{E}(\|\mathbf{X} - \hat{\mathbf{x}}\|_2^2). \tag{2.10}$$

From (6) and (7) we obtain that this evaluation is given using the Bayesian approach. In this way we can reach information of an unknown variable X, drawn by the probability density function, $f_X(x)$, which is a prior distribution, using a related random variable Y. Therefore, having the value y of Y, we are able to find the posterior distribution of X, $f_{X|Y}(x|y)$, which is found through the Bayes' formula:

$$f_{X|Y}(x|y) = \frac{f_{Y|X}(y|x)\ f_X(x)}{f_Y(y)}, \tag{2.11}$$

and which contains all the informations we know about X. Starting from that we can find the estimate of X. It is given by $\hat{x} = g(Y)$, then to find the best one we have to minimize the expected mean squared error

$$MSE = \mathbb{E}((X - \hat{x})^2 \mid y) = \mathbb{E}((X - g(Y))^2 \mid y). \tag{2.12}$$

Calling a our estimate, we want to minimize the function

$$h(a) = \mathbb{E}((X - a)^2 \mid y) = \mathbb{E}(X^2 \mid y) - 2a\mathbb{E}(X \mid y) + a^2, \tag{2.13}$$

and deriving respect to a, and nulling it we obtain that the minimum is reached by

$$\hat{x}_{MMSE} = \mathbb{E}(X \mid y), \tag{2.14}$$

which is the Bayes' estimate of X. Translating the above discussion into our case, we want to reach information of an unknown image $\mathbf{X}$, drawn by the probability density function, $f_{\mathbf{X}}(x)$, using its measurement $\mathbf{Y}$ related to it through the conditional probability $f_{\mathbf{Y}|\mathbf{X}}(y|x)$. Starting from that we can find the best estimate minimizing

$$MSE = \mathbb{E}(\|\mathbf{X} - g(\mathbf{Y})\|_2^2 \mid y) = \int \|\mathbf{X} - g(\mathbf{Y})\|_2^2 \, f_{\mathbf{X}|\mathbf{Y}}(x|y) \, d\mathbf{X} \tag{2.15}$$

Thus, we have to derive it respect to $g(\mathbf{Y})$ and null it, finding

$$g_{MMSE}(y) = \int_{\mathbf{X}} \mathbf{X} \, f_{\mathbf{X}|\mathbf{Y}}(x|y) \, d\mathbf{X} = \mathbb{E}(\mathbf{X}|\mathbf{Y}), \tag{2.16}$$

where the last equality come from 2.14 and the above discussion, meaning that the expectation is seen as the mean of the posterior conditioned probability. Finally using Bayes' formula 2.11, we are able to compute $f_{\mathbf{X}|\mathbf{Y}}(x|y)$, and putting it in 2.16, we have found the minimum value

$$\hat{\mathbf{x}}_{MMSE} = \mathbb{E}(\mathbf{X}|\mathbf{Y}). \tag{2.17}$$

Therefore if we want to know the MSE for this estimator we have to compute

$$MSE = \mathbb{E}(\|\mathbf{X} - \mathbb{E}(\mathbf{X}|\mathbf{Y})\|_2^2) = \frac{1}{N} \sum (\mathbf{X} - \mathbb{E}(\mathbf{X}|\mathbf{Y}))^2, \tag{2.18}$$

which gives us the lower bound of MSE for the image $\mathbf{X}$, and we can use it as a stopping criterion: when for an image the MSE between $\mathbf{X}$ and its denoised version is near that value, then we have reached a good denoised image. Finding $\hat{\mathbf{X}}_{MMSE}$, however, is nearly impossible with the built network, in fact it is an ideal computation. This is relevant in terms of the noise; if we want to consider also the structure of the image, without losing important contents, we have to make other considerations.

2.5.2 Perception-Distortion Tradeoff

In (7) is discussed if, even reaching the maximum noise removal, it can result in blurry images, compromising perceptual quality of the image. Has, in fact, been proved that there exists a "perception-distortion tradeoff", where in our case distortion means noise; this tradeoff implies that optimizing one results in deteriorating the other (see figure 2.7). With this knowledge we can then improve our research stating that trying only to minimize the MSE between the clean image and its denoised version is useless if we want to restore the image wisely. For that reason it is necessary to consider a good mean in minimizing distortion getting good visual quality. First of all we will follow other roads, for example substituting the MSE loss function with some alternatives that consider the structure of the image. We can ask why this can be a wise approach, so before describing the measures, we will see that the answer relies on the cited

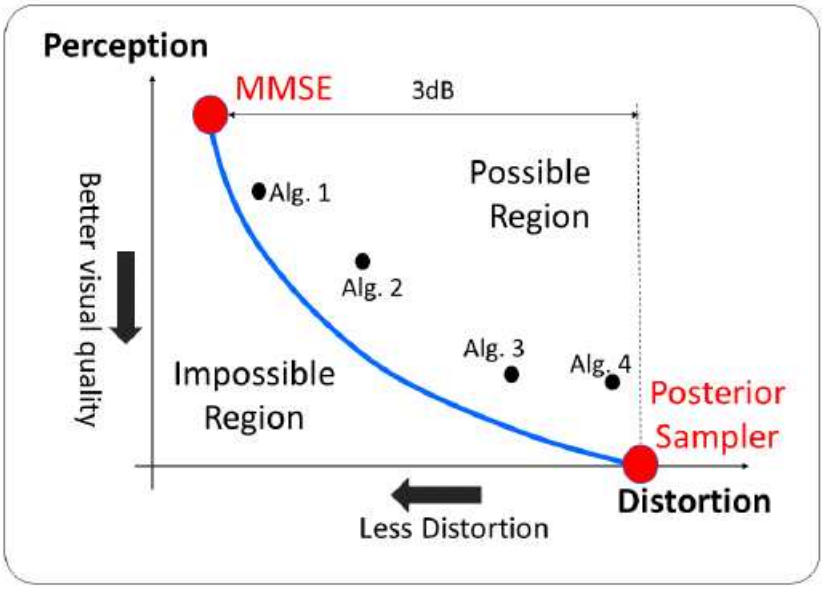

Figure 2.7

tradeoff. As we have seen before, we can draw an ideal image $\mathbf{X}$ from the image manifold, represented by the PDF $f_{\mathbf{X}}(x)$, measure the noised image relating it with a conditional probability $f_{\mathbf{X}|\mathbf{Y}}(x|y)$ and using it, we can get an expectation $\hat{\mathbf{X}}$ of it. We have also seen that to evaluate how well the image has been denoised, we need a distance function $\Delta(\mathbf{X}, \hat{\mathbf{X}})$, and to measure the distortion of the estimator we average between the values using $\mathbb{E}(\Delta(\mathbf{X}, \hat{\mathbf{X}}))$. On the other hand to evaluate the perceptual quality of an estimator is used a divergence between the PDFs of the ideal image and the reconstructed one $d(f_{\mathbf{X}}, f_{\hat{\mathbf{X}}})$. The best perceptual quality is attained when $f_{\mathbf{X}} = f_{\hat{\mathbf{X}}}$, meaning when the output image belongs to the original image manifold. Following the consideration described above we can state that generally $f_{\mathbf{X}} \nsim f_{\hat{\mathbf{X}}_{MMSE}}$, implying that low distortion does not implies good perceptual quality. We can observe that this represent a Pareto frontier, indeed in 2.7 an increasing in distortion influences positively the perception quality, and viceversa. We can see that defining

$$P(D) = \min_{f_{\hat{\mathbf{X}}|\mathbf{Y}}} d(f_{\mathbf{X}}, f_{\hat{\mathbf{X}}}) \quad s.t. \quad \mathbb{E}(\Delta(\mathbf{X}, \hat{\mathbf{X}})) \leq D, \tag{2.19}$$

where $P(D)$ is the minimal deviation between the PDFs $f_{\mathbf{X}}$ and $f_{\hat{\mathbf{X}}}$ that can be attained by an estimator with distortion D. In (8) is stated the following theorem, which assume that the perception-distortion tradeoff does not depend on $\Delta(\cdot, \cdot)$, so it exists for every distortion measure.

Teorema 2.5.1 (The perception-distortion tradeoff)**.** *Following the previous*

assumptions on distortion and perceptual quality, if $d(f_{\boldsymbol{X}}, f_{\hat{\boldsymbol{X}}})$ is convex in its second argument, then the perception-distortion function $P(D)$ in 2.19 is

1. *monotonically non-increasing;*

2. *convex.*

However, this does not imply that all the distortion measure have the same perception-distortion function, indeed the tradeoff can be better for distortion measures that control structural similarity between images.

For that reason, following (5) we will focus on a mix function which combine the multiscale version of SSIM and the Mean Absolute Error (MAE) and inspired by that and (9) we will consider the combination of SSIM with MSE.

2.5.3 Multi-Scale Structural Similarity combined to Mean Absolute Error

In (5), is proposed the multiscale version of 2.8,

$$MS - SSIM(\mathbf{X}, \hat{\mathbf{X}}) = l_M(\mathbf{X}, \hat{\mathbf{X}}) \cdot \prod_{j=1}^{M} cs_j(\mathbf{X}, \hat{\mathbf{X}}). \tag{2.20}$$

If we want to consider local structures of images we can consider a patch P of the image, in which pixels are denoted as p. Going more in depth in 2.8, we have

$$l(\mathbf{X}, \hat{\mathbf{X}}) = \frac{2\mu_{\mathbf{X}}\mu_{\hat{\mathbf{X}}} + C_1}{\mu_{\mathbf{X}}^2 + \mu_{\hat{\mathbf{X}}}^2 + C_1}, \tag{2.21}$$

where the mean is computed respect to the central pixel. So it results

$$\mu_x = \sum_{i=1}^{N} G_{\sigma_G}(x_i)\ x_i, \tag{2.22}$$

with x the image $\mathbf{X}$ or $\hat{\mathbf{X}}$, and x_i the pixels in the related image. This is the mean computed with a $2D$ Gaussian blur

$$G_{\sigma_G}(x) = \frac{1}{\sqrt{2\pi}\sigma_G} e^{-\frac{r_x^2 + c_x^2}{2\sigma_G^2}} \tag{2.23}$$

with standard deviation σ_G and r_x, c_x the coordinates of pixel x from the central pixel, meaning that the one nearer to the origin has more impact on the function. Therefore, $l(\mathbf{X}, \hat{\mathbf{X}})$ compares the local luminance between two images using mean intensity and brightness weighting the pixels inside a patch. The constant C_1 in 2.21 controls the stability of denominator. Furthermore, the term $cs(\mathbf{X}, \hat{\mathbf{X}})$ combines the formulas of contrast comparison and structural

similarity multiplying them. The first is determined using standard deviation computed with the Gaussian smoothing

$$\sigma_x = \left(\sum_{i=1}^{N} G_{\sigma_G}(x_i)(x_i - \mu_x)^2 \right)^{\frac{1}{2}} \tag{2.24}$$

with x as before equal to $\mathbf{X}$ or $\hat{\mathbf{X}}$, and a stabilizing constant C_2,

$$c(\mathbf{X}, \hat{\mathbf{X}}) = \frac{2\sigma_{\mathbf{X}}\sigma_{\hat{\mathbf{X}}} + C_2}{\sigma_{\mathbf{X}}^2 + \sigma_{\hat{\mathbf{X}}}^2 + C_2}. \tag{2.25}$$

The structural comparison term works with luminance subtraction and variance normalization of the images and a constant which in this case is posed equal to $C_3 = \frac{C_2}{2}$

$$s(\mathbf{X}, \hat{\mathbf{X}}) = \frac{\sigma_{\mathbf{X}\hat{\mathbf{X}}} + C_3}{\sigma_{\mathbf{X}}\sigma_{\hat{\mathbf{X}}} + C_3}, \tag{2.26}$$

where

$$\sigma_{xy} = \sum_{i=1}^{N} G_{\sigma_G}(x_i)(x_i - \mu_x)(y_i - \mu_y), \tag{2.27}$$

with x and y denoting respectively $\mathbf{X}$ and $\hat{\mathbf{X}}$. All this formulas are then transposed in multi-scale terms, which is done using a dyadic Pyramid of M levels in which an image is processed by smoothing and subsampling through a Gaussian filter, reducing noise and emphasizing important structures of the image. The luminance comparison 2.21 is made only at last scale M, while contrast and structure comparison 2.25 and 2.26 are computed at every scale $j = 1, \dots M$. The related loss function is defined as

$$\varepsilon^{MS-SSIM}(\mathbf{X}, \hat{\mathbf{X}}) = 1 - MS - SSIM(\mathbf{X}, \hat{\mathbf{X}}), \tag{2.28}$$

comparing all pixels p for patch P. The multi-scale method is a convenient way to incorporate image details at different resolutions, which permits to achieve better results than SSIM. Indeed, due to its multiscale nature, MS-SSIM solves the issue of noise better near edges, where the change of color (in grayscale) is neat, than that in flat areas, where it can cause shift in brightness and colors. However, still preserves contrast at high frequencies better than the other loss functions. On the other hand is considered the MAE

$$\varepsilon^{MAE} = \frac{1}{N} \sum_{i} |X_i - \hat{X}_i|, \tag{2.29}$$

computed pixel-wise, where X_i is the i-th element of the image $\mathbf{X}$. It preserves colors and luminance, because the error is weighted in the same way for every pixel, so does not depend on the image patch structures, however does not produce the same contrast as MS-SSIM. From that reason the MIX loss function is proposed

$$\varepsilon^{MIX} = \alpha \cdot \varepsilon^{MS-SSIM} + (1 - \alpha) \cdot G_{\sigma_G^M} \cdot \varepsilon^{MAE}, \tag{2.30}$$

where α gives the right weight to the two losses to capture the best characteristics of both. For the $MS-SSIM$, instead of computing the M levels of Pyramid, in (5) it is approximated using M different values of σ_G^i at level i, each one being the half of the previous. This is done assuming $cs_i = G_{\sigma_G^i} \cdot cs_0(\tilde{p})$, where the Gaussian filter is centered at pixel $\tilde{p}$. The point-wise multiplication between ε^{MAE} and $G_{\sigma_G^M}$ is set because MS-SSIM control the error for pixel p based on its distance with the central pixel $\tilde{p}$. From the above we can easily conclude that the benefit of the MIX loss can be various in denoising and preserving the structure of the image. However for our purpose the discussion can be simplified: in fact working with gray-scale images and white Gaussian noise it is not necessary to get a multiscale decomposition of the image so to still preserve structural content of the image we can consider the simpler non multiscale version, SSIM. Moreover we can state that for our use is sufficient luminance, contrast and structure similarity in the global image as the noise does not strictly depend from local structures, while using the Gaussian blur permits to give different weights to the local structures of the images, from central pixel, to which is given more importance, to periferical pixels which are blurred. On the other hand as the simplest version analyzes the mean luminance, contrast and structure similarity on the global image, to avoid for example change of brightness we should also control the content pixel by pixel, so basically we can combine the SSIM with the already used MSE.

2.5.4 Mean Squared Error combined with Structural Similarity Index Measure

The SSIM loss function is computed as the inverse of 2.8, meaning that minimizing the loss, the SSIM index is maximized. In formulas we have

$$\varepsilon^{SSIM}(\mathbf{X}, \hat{\mathbf{X}}) = 1 - SSIM(\mathbf{X}, \hat{\mathbf{X}}) = 1 - l(\mathbf{X}, \hat{\mathbf{X}}) \cdot cs(\mathbf{X}, \hat{\mathbf{X}}), \tag{2.31}$$

where luminance l and contrast-structure cs similarity are defined as follow.

$$l(\mathbf{X}, \hat{\mathbf{X}}) = \frac{2\mu_{\mathbf{X}}\mu_{\hat{\mathbf{X}}} + C_1}{\mu_{\mathbf{X}}^2 + \mu_{\hat{\mathbf{X}}}^2 + C_1}, \tag{2.32}$$

where

$$\mu_x = \frac{1}{N}\sum_{i=1}^{N} x_i \tag{2.33}$$

is the mean of image x,

$$cs(\mathbf{X}, \hat{\mathbf{X}}) = c(\mathbf{X}, \hat{\mathbf{X}}) \cdot s(\mathbf{X}, \hat{\mathbf{X}}) = \frac{2\sigma_{\mathbf{X}}\sigma_{\hat{\mathbf{X}}} + C_2}{\sigma_{\mathbf{X}}^2 + \sigma_{\hat{\mathbf{X}}}^2 + C_2} \cdot \frac{\sigma_{\mathbf{X}\hat{\mathbf{X}}} + C_3}{\sigma_{\mathbf{X}}\sigma_{\hat{\mathbf{X}}} + C_3}, \tag{2.34}$$

where

$$\sigma_x{}^2 = \frac{1}{N}\sum_{i=1}^{N}(x_i - \mu_x)^2 \tag{2.35}$$

is the variance and

$$\sigma_{xy} = \frac{1}{N}\sum_{i=1}^{N}(x_i - \mu_x)(y_i - \mu_y) \tag{2.36}$$

the covariance of the ground truth image and the denoised one, and all the summations are done pixel-wise. Using $C_3 = \frac{C_2}{2}$ as in (5), cs become

$$cs(\mathbf{X}, \hat{\mathbf{X}}) = \frac{2\sigma_{\mathbf{X}\hat{\mathbf{X}}} + C_2}{\sigma^2_{\mathbf{X}} + \sigma^2_{\hat{\mathbf{X}}} + C_2} \tag{2.37}$$

Following the idea in (5) and (9) we consider the following loss function

$$\varepsilon^{MIX} = \alpha \cdot \varepsilon^{SSIM} + (1 - \alpha) \cdot \varepsilon^{MSE}, \tag{2.38}$$

As mentioned before MSE compares images pixel by pixel, and due of the square difference it penalizes larger errors and in an equivalent way more tolerant to small errors meaning that it results in smoothing the image removing noise, but it can results in blurring them. For example in flat regions it strongly attenuates the noise, but can results in splotchy artifacts. On the other hand the sharpness of the edges is well preserved when the contrast is neat, however where the noise does not affect the image content in a neat way, meaning the error is small it tends to not to consider it. On the other hand the SSIM controls details and textures which can be lost by MSE. However in flat region colors can be shifted or brightness can be changed by SSIM, but MSE preserve it because compares singular pixels. In conclusion if the goal is to get a better result for our implemented net, removing the noise but still preserve the images contents, this is the path to follow, also because more it is more understandable for the use in backpropagation if we want to analyze an application.

If we want to train our network with this new loss function, we will then need to compute its gradient respect to the network weights $\Theta^{(t)}$ and choose α in a proper way. For the gradients, most of the computations remain the same as in 2.4.3 for the chain rule, a part from the weights in the loss layer i.e. the gradients respect to a^k_T, $\mathbf{D}^T$ and $\mathbf{w}$, which we will now call in general as $\Theta^{(T)}$. In fact, we will compute

$$\bigtriangledown_{\Theta^{(T)}}\varepsilon^{MIX} = \alpha \cdot \bigtriangledown_{\Theta^{(T)}}\varepsilon^{SSIM} + (1 - \alpha) \cdot \bigtriangledown_{\Theta^{(T)}}\varepsilon^{MSE}. \tag{2.39}$$

More specifically we will only need to compute $\bigtriangledown_{\Theta^{(T)}}\varepsilon^{SSIM}$, in particular for the chain rule $\frac{\partial\varepsilon^{SSIM}(\mathbf{X},\hat{\mathbf{X}})}{\partial\Theta^{(T)}(i,j)} = \sum_r \frac{\partial\varepsilon^{SSIM}(\mathbf{X},\hat{\mathbf{X}})}{\partial\hat{\mathbf{X}}_r} \cdot \frac{\partial\hat{\mathbf{X}}_r}{\partial\Theta^{(T)}(i,j)}$, where the first term is computed as follow for each r,

$$\frac{\partial\varepsilon^{SSIM}(\mathbf{X},\hat{\mathbf{X}})}{\partial\hat{\mathbf{X}}_r} = -\frac{\partial l(\mathbf{X},\hat{\mathbf{X}})}{\partial\hat{\mathbf{X}}_r} \cdot cs(\mathbf{X},\hat{\mathbf{X}}) - l(\mathbf{X},\hat{\mathbf{X}}) \cdot \frac{\partial cs(\mathbf{X},\hat{\mathbf{X}})}{\partial\hat{\mathbf{X}}_r} \tag{2.40}$$

while the second has already been computed. In the end we will only need $\frac{\partial\mu_{\hat{\mathbf{X}}}}{\partial\hat{\mathbf{X}}_r}$, $\frac{\partial\sigma_{\hat{\mathbf{X}}}}{\partial\hat{\mathbf{X}}_r}$ and $\frac{\partial\sigma_{\mathbf{X}\hat{\mathbf{X}}}}{\partial\hat{\mathbf{X}}_r}$ which are the only terms in 2.32 and 2.37 that depends on $\hat{\mathbf{X}}_r$. Finally we find

$$\frac{\partial\mu_{\hat{\mathbf{X}}}}{\partial\hat{\mathbf{X}}_r} = \frac{1}{N} \quad \forall r$$

$$\frac{\partial \sigma_{\mathbf{X}\hat{\mathbf{X}}}}{\partial \hat{\mathbf{X}}_r} = \frac{1}{N}\left(X_r - \mu_{\mathbf{X}}\right) \quad \forall r$$

$$\frac{\partial \sigma_{\hat{\mathbf{X}}}}{\partial \hat{\mathbf{X}}_r} = \frac{1}{\sigma_{\hat{\mathbf{X}}} N}\left(\hat{X}_r - \mu_{\hat{\mathbf{X}}}\right) \quad \forall r.$$

Denoting

$$\bar{\mu}_{\mathbf{X}} = \mu_{\mathbf{X}} \cdot ONES(N)$$

with point-wise multiplication, and writing

$$\partial l(\mathbf{X}, \hat{\mathbf{X}}) = \frac{\bar{\mu}_{\mathbf{X}} - \bar{\mu}_{\hat{\mathbf{X}}} \cdot l(\mathbf{X}, \hat{\mathbf{X}})}{\mu_{\mathbf{X}}^2 - \mu_{\hat{\mathbf{X}}}^2 + C_1}$$

and

$$\partial cs(\mathbf{X}, \hat{\mathbf{X}}) = \frac{(\mathbf{X} - \bar{\mu}_{\mathbf{X}}) - (\hat{\mathbf{X}} - \bar{\mu}_{\hat{\mathbf{X}}}) \cdot cs(\mathbf{X}, \hat{\mathbf{X}})}{\sigma_{\mathbf{X}}^2 - \sigma_{\hat{\mathbf{X}}}^2 + C_2}$$

we can conclude

$$\partial \varepsilon^{SSIM}(\mathbf{X}, \hat{\mathbf{X}}) = -\frac{2}{N}\left(cs(\mathbf{X}, \hat{\mathbf{X}}) \cdot \partial l(\mathbf{X}, \hat{\mathbf{X}}) + l(\mathbf{X}, \hat{\mathbf{X}}) \cdot \partial cs(\mathbf{X}, \hat{\mathbf{X}})\right).$$

In conclusion we find:

- **Gradient with respect to the sparse representation a_T^k**:

$$\bigtriangledown_{a_T^k} \varepsilon^{SSIM} = -\frac{2}{N} \mathbf{D}^{(T)^T} \left(\mathbf{R}_k^T \cdot \frac{\partial \varepsilon^{SSIM}(\mathbf{X}, \hat{\mathbf{X}})}{\sum_k \mathbf{R}_k^T \mathbf{w}} \circ \mathbf{w}\right),$$

- **Gradient with respect to the dictionary $\mathbf{D}^{(T)}$**:

$$\bigtriangledown_{\mathbf{D}^{(T)}} \varepsilon^{SSIM} = -\frac{2}{N} \sum_k \left(\mathbf{R}_k \cdot \frac{\partial \varepsilon^{SSIM}(\mathbf{X}, \hat{\mathbf{X}})}{\sum_k \mathbf{R}_k^T \mathbf{w}} \circ \mathbf{w}\right) a_T^{k\,T},$$

- **Gradient with respect to the weight $\mathbf{w}$**:

$$\bigtriangledown_{\mathbf{w}} \varepsilon^{SSIM} = -\frac{2}{N} \sum_k \left[\left[\left(\mathbf{R}_k \cdot \frac{\partial \varepsilon^{SSIM}(\mathbf{X}, \hat{\mathbf{X}})}{\sum_k \mathbf{R}_k^T \mathbf{w}}\right) \circ \hat{\mathbf{x}}_k\right] - \mathbf{R}_k \cdot \left[\frac{\partial \varepsilon^{SSIM}(\mathbf{X}, \hat{\mathbf{X}})}{(\sum_k \mathbf{R}_k^T \mathbf{w})^2} \circ \left(\sum_k \mathbf{R}_k^T (\hat{\mathbf{x}_k} \circ \mathbf{w})\right.\right.\right.$$

Starting from that, with the same computations of 2.4.3, we can backpropagate the output with the new mix loss function.

2.5.5 Test image quality

To check the quality of reconstructed images the Peak signal-to-noise ratio (PSNR) is used. It depends on MSE and it is defined as

$$PSNR = 10 \cdot \log_{10} \frac{\max_i^2 \hat{\mathbf{X}}}{\varepsilon^{MSE}(\mathbf{X}, \hat{\mathbf{X}})},$$

for an output image $\hat{\mathbf{X}}$ and $\max_i \hat{\mathbf{X}}$ the maximum possible pixel value in the image. It has been used in various image denoising tasks to check quality of processed images, for example in (1) it is used to compare the performance of deep K-SVD denoiser to the near state-of-the-art denoisers. The PSNR value approaches infinity as the MSE approaches zero; this shows that a higher PSNR value provides a higher image quality, while a small value of the PSNR implies high numerical differences between images. Knowing that the PSNR depends on the MSE, based on what we have discussed before it is able to check the quality of the image depending on how well the noise has been removed. However to check quality depending on structural content of the image it is wise to use SSIM index 2.8 which takes values between $[0, 1]$. As we can see in the definition 2.8 the two images are equal if SSIM is 1. Furthermore in (16) we can find a relationship between the two metrics and compute the related quality. In fact,

$$PSNR = 10 \cdot \log_{10} \left[\frac{2\sigma_{\mathbf{X}\hat{\mathbf{X}}}(l(\mathbf{X}, \hat{\mathbf{X}}) - SSIM)}{\max_i^2 \hat{\mathbf{X}} \cdot SSIM} + \left(\frac{\mu_{\mathbf{X}} - \mu_{\hat{\mathbf{X}}}}{\max_i \hat{\mathbf{X}}} \right)^2 \right].$$

This follows rewriting 2.9 as

$$\varepsilon^{MSE}(\mathbf{X}, \hat{\mathbf{X}}) = \sigma_{\mathbf{X}}^2 + \sigma_{\hat{\mathbf{X}}}^2 - 2\sigma_{\mathbf{X}\hat{\mathbf{X}}} + (\mu_{\mathbf{X}} - \mu_{\hat{\mathbf{X}}})^2$$

This will be explained better in the next chapter, where we will analyze the implemented code structure and how the network perform with examples.

Another interesting path to follow is inspired by (14), that beyond taking care of the error using a likelihood function to control how the algorithm has to remove the noise, it gives importance to the residual given by the original image and the output image with the noisy image. It should have a AGWN nature, in fact the discrepancy between the original image and its noisy version is the noise $\mathbf{V}$, while the discrepancy between the denoised image and the noisy is a residual $\mathbf{R}$ that during training should get closer and closer to the given noise. If it is not of this nature other errors are maybe involved, meaning that the network is not operating in the right way, still leaving some signal components. In the paper the Normalized Cumulative Periodogram (NCP), or Bartlett test, is used to verify if the residual is white noise.

Therefore, we first define the NCP and then compute the theoretical NCP applied to $\mathbf{V}$ and the empirical one applied to the residual $\mathbf{R}$ to compare them and verify that the second is something like the first. The 2D periodogram, or

power spectrum, of an $M \times N$ image $\mathbf{X}$ is a $p \times q$ matrix $\mathbf{P}$, where its elements are defined as

$$P_{kl} = |\text{dft2}(\mathbf{X})_{kl}|^2, \qquad k = 1, \ldots, p, \quad l = 1, \ldots, q.$$

with

$$\text{dft2}(\mathbf{X})_{kl} = \sum_{m=1}^{M} \sum_{n=1}^{N} X_{mn} e^{-2\pi i \left(\frac{(k-1)(m-1)}{M} + \frac{(l-1)(n-1)}{N} \right)} = \left[F_2^T X F_1 \right]_{kl},$$

the 2D Discrete Fourier Transform (DFT), defining

$$[F_2]_{mk} = e^{-2\pi i \left(\frac{(k-1)(m-1)}{M} \right)} \tag{2.41}$$

the transform along the columns and

$$[F_1]_{nl} = e^{-2\pi i \left(\frac{(l-1)(n-1)}{N} \right)} \tag{2.42}$$

the transform along the rows. The NCP for the image $\mathbf{X}$ is then defined as the $c(\mathbf{X})$ vector of length $p \cdot q + 1$ with components

$$c(\mathbf{X})_k = \frac{\|\hat{p}(2 : k+1)\|_1}{\|\hat{p}(2 : p \cdot q)\|_1}, \qquad k = 1, \ldots, \ pq - 1, \tag{2.43}$$

where

$$\hat{p} = \Pi \ \text{vec}(\mathbf{P})$$

obtained multiplying the vector which stacks all the columns of $\mathbf{P}$ into a vector of length $p \cdot q$ with a permutation matrix Π such that $\hat{p}$ holds all the power spectrum elements in order of increasing spatial frequency.

Recalling that our problem is of type 1.1, where $\mathbf{V}$ is white noise, we now discuss the NCP test and how to use it for our problem. More precisely considering 1.2 and the 2D DFT of $\mathbf{V}$, then denoting $dft2(\mathbf{V}) = \mathbf{F}_2^T \mathbf{V} \mathbf{F}_1$ and knowing that $\mathbf{F}_1$ and $\mathbf{F}_2$ are unitary matrices and computing $\text{dft2}(\mathbf{V}) \ \text{dft2}(\mathbf{V})^T = \mathbf{F}_2^T \ \mathbf{V}\mathbf{V}^T \mathbf{F}_2$, it results

$$\begin{aligned} \text{Cov}(\text{dft2}(\mathbf{V})) = \mathbb{E}(\text{dft2}(\mathbf{V}) \ \text{dft2}(\mathbf{V})^T) = \mathbf{F}_2^T \mathbb{E}(\mathbf{V}\mathbf{V}^T) \mathbf{F}_2 = \\ = \mathbf{F}_2^T \, \text{Cov}(\mathbf{V}) \mathbf{F}_2 = \sigma^2 \mathbf{F}_2^T \mathbf{F}_2 = \sigma^2 \mathbb{I} \end{aligned} \tag{2.44}$$

From here we know that the diagonal of the covariance matrix, which in this case is equal to the variance of the signal, is constant and it is equal to the expected value of the scaled power spectrum, meaning that all the frequencies have the same expectation. This is how the white noise behaves, deducing that the expected values of the NCP lies in a straight line between $(0, 0)$ and $(p \cdot q, 1)$. If we want to test if the residual is white noise as in (14), then we have to check if the NCP of $\mathbf{R}$ lie within the Kolmogorov-Smirnoff limits of the straight line. Finally if we want to track if the residual is of white noise nature could be interesting to compute the deviation of the NCP of the residual from

the straight line and how it varies for the different computed residuals. We can do that computing the following for the related residual:

$$\mathcal{N} = \|\mathbf{s} - c(\mathbf{R})\|_1, \tag{2.45}$$

where the components of $\mathbf{s}$ are $s_i = \frac{i}{p \cdot q - 1}$.

The question is how can we get advantage of these considerations to improve our study. On the one hand we will apply the test to the residuals related to the different networks trained with the different loss functions to verify if our previous theoretical examinations are relevant. For example, as we will illustrate in the next chapter, we can compare how the NCP defined in 2.43 and 2.45 varies in our different cases depending on the parameter α of the MIX loss function, and compare those values with the results related to the MSE loss function.

Chapter 3

Applications

In this chapter we will discuss the construction of our `NumPy` network, justifying some choices and discussing the results we have found. Following the architecture described in section 2.2, we test it using some images, disjoint from the ones we used during the training. The images in our dataset are the same used in the architecture (3) to which (1) is referred, for a comparison. It consists on 500 images in gray scale of dimensions 481×321 or 321×481. As outlined in 2.4, this dataset is partitioned into two separate subsets: 432 for training, 68 for testing. In order to have a larger number of images for training and faster the process, all 432 training images are cropped to dimensions 128×128. This augmentation yields a dataset comprising nearly $70,000$ images, which are subsequently normalized to fall within the range of values $[-1, 1]$. Throughout the training phase, our network randomly selects images one at a time, matching the batch size, and introduces AWGN. Subsequently, it processes the resulting noised image to produce the denoised cropped image, as depicted in 2.4.2. The output is then compared to the original cropped image using a chosen loss function; the value is then saved to have control on the performance the training process. Finally the output is back-propagated as in 2.4.3 in order to optimize the weights of the network. All these steps are applied for each cropped image in the dataset until the training is finished. A crucial question is decide when the training process can be stopped avoiding over-fitting or under-fitting of data. In our case we trained the algorithm for 20 hours as it is done in (1) saving the last weights update.
In order to test the algorithm, we first load the trained weights and use various types of full-dimensional images as input to assess the effectiveness of noise removal and image restoration across diverse data. In the following sections, we will present the algorithm, its settings, and the resulting outputs, providing empirical insights into the theoretical discussions presented in previous chapters.

3.0.1 Parameters Setting

The parameters to be set in the network are discussed below; given a cropped image the function 1 in Appendix 3.0.2 adds noise to it, which consist on a matrix of dimensions equal to the image, whose entries are picked randomly following a uniform distribution, and then multiplied element-wise with the chosen level of noise σ. Moreover, in our implemented network we decompose the input cropped image $\mathbf{Y}$ into overlapping patches $\mathbf{y}_k$ of dimensions $\sqrt{p} \times \sqrt{p}$, selected by an operator $\mathbf{R}_k$ for forward propagation. The construction of the operator is shown in 2. Furthermore, during the sparse coding stage, an important step lies in determining the number of iterations T of the LISTA algorithm, as depicted in 4. We choose

- $\sigma = 25$ as noise level;
- $\sqrt{p} = 8$ as dimension of the patches;
- $T = 7$ as number of layers in the sparse code.

We tested also with less layers, $T = 5$ but the training results less efficient. Moreover, the initialization of the learnable weights of the network is illustrated in 6. For these we have to set

- the dimensions of dictionary D: $8^2 \times 16^2$;
- the dimension of the MLP nodes depends on p;
- the activation function of last node is chosen as ReLU;
- c is a scalar, the squared spectral norm of D;
- the weight w has the dimensions of the patches.

Finally for back-propagation we choose

- as batch size one cropped image, but it is possible to do the training using more images;
- ADAM as optimizer with $\beta_1 = 0.9$, $\beta_2 = 0.999$, $epsilon = 10^{-16}$ and learning rate $lr = 10^{-4}$, but also the SGD optimizer is implemented;
- MSE as loss function, but we trained the net also with the MIX loss function 2.5.4 to verify the theoretical considerations.

For the last point, the choice of the loss function, we trained the network with MSE loss function and MIX function with several values of α, to check if the theoretical considerations in 2.5.4 corresponds to empirical computations. To implement the MIX loss function we followed (4), where the stabilizing constants in 2.32 and 2.37 are defined as $C_1 = (LK_1)^2$ and $C_2 = (LK_2)^2$, with $K_1, K_2 \ll 1$ and L the dynamic range for pixels values, which is 255 for grayscale images, but 1 for normalized images as the data in our dataset. In

(4) $K_1 = 0.01$ and $K_2 = 0.03$ meaning that $C_1 = 10^{-4}$ and $C_2 = 9 \cdot 10^{-4}$ as we choose. We utilized a range of values for α in our experiments, specifically $\alpha = 0, 0.2, 0.4, 0.6, 0.69, 0.7, 0.71, 0.8$. The initial value of $\alpha = 0$ served as a baseline to verify the implementation accuracy of the MIX loss function by comparing it against MSE. Subsequent values were chosen to assess if the impact of increasing weights on structural similarity could result in an improvement in the performance. The decision to focus values around 0.7 lies in the fact that we begin our analysis with a value of $\alpha = 0.6$, as suggested in (5) and (9), in order to test the network and check if it could have any improvement. Furthermore, we experimented with values around this range and we could observe that $\alpha = 0.7$ yielded the optimal balance between denoising efficacy and visual quality. Finally, we tested all these deductions using NCP for the behaviour of the residual, PSNR to check the performance of denoising in the output image, and to have a comparison with other denoisers; in the same way we computed SSIM, to check structural similarity between the denoised image and the input image. We will discuss better this last point, with concrete examples, in the following section.

3.0.2 Output examples

To begin our analysis we first discuss the results found training the network with the MSE loss function; the denoiser will be from now on denoted as $\mathcal{D}_{MSE}$ for simplicity. We set the AGWN with $\sigma = 25$ as in (1). In figure 3.1 we

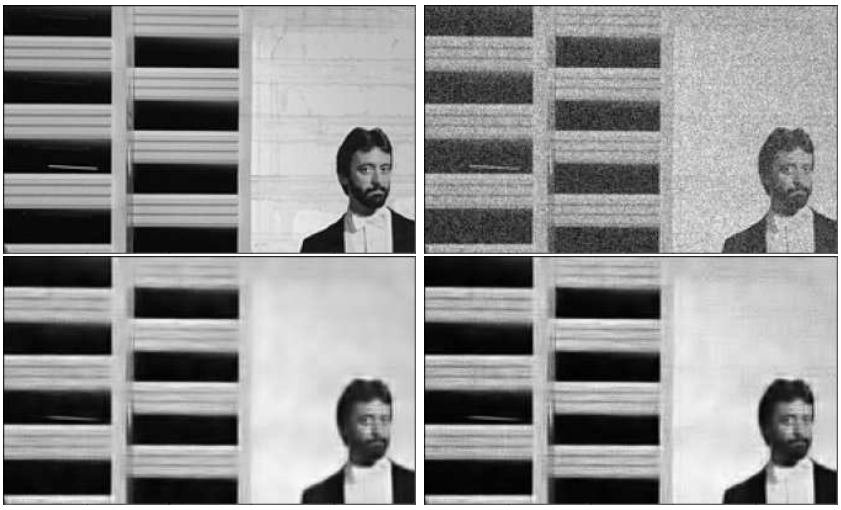

Figure 3.1: Denoiser $\mathcal{D}_{MSE}$ applied to the input image with values $MSE \sim 0.006$, $PSNR \sim 22$: (a) Top left: Original clean image, (b) Top right: Image with added noise, (c) Bottom left: Image denoised using $\mathcal{D}_{MSE}$, (d) Bottom right: Image denoised using $\mathcal{D}_{MIx}^{0.7}$.

can see one of test images to which has been added AGWN. The bottom left

image has been denoised using $\mathcal{D}_{MSE}$, while in the right the image has been denoised using $\mathcal{D}^{0.7}_{MIX}$, where with $\mathcal{D}^{\alpha}_{MIX}$ we denote the denoiser trained using 2.5.4, with α the regularization parameter. Here we can notice that, where the contrast between colors is neat, the image has been denoised and restored in a good way, however some details in uniform surfaces , such as the bricks on the wall or some details on the shirt of the man are lost. We can make almost

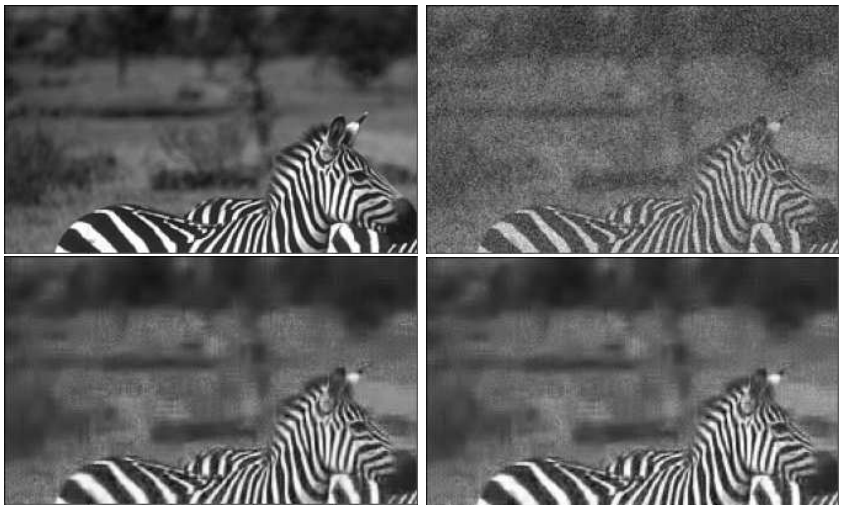

Figure 3.2: Denoiser $\mathcal{D}_{MSE}$ applied to the input image with values $MSE \sim 0.009$, $PSNR \sim 20,5$: (a) Top left: Original clean image, (b) Top right: Image with added noise, (c) Bottom left: Image denoised using $\mathcal{D}_{MSE}$, (d) Bottom right: Image denoised using $\mathcal{D}^{0.7}_{MIX}$.

the same considerations for image 3.2, where the details of the zebras are well restored; however, the background, which is almost blurry in the original image, in the denoised version is not well detailed and presents some splotches. When analyzing a more uniform image, whether it's bright or dark as depicted in 3.3, we observe a loss of details. This observation leads us to conclude that our algorithm performs better for images with high frequencies. However, in the case of low frequencies, while the algorithm effectively removes noise, it struggles to fully restore the image, often resulting in blurry areas. If we analyze the MSE values, we can confirm what we have discussed in Section 2.5. In fact a low value of MSE corresponds to an higher value of PSNR which is related to the quality of the image in terms of noise. However if we observe the denoised image, it is not well reconstructed in terms of structure and contents. We will below compare $\mathcal{D}_{MSE}$ and $\mathcal{D}^{\alpha}_{MIX}$; for the second, we choose some values of α in $[0, 1]$, as explained in the previous section.
We computed the NCP for all the considered denoisers, to test which residual is nearest to the added Gaussian white noise together with the deviation from the straight line 2.45:

$$\mathcal{N} = \|\mathbf{s} - c(\mathbf{R}_{\alpha})\|_1. \tag{3.1}$$

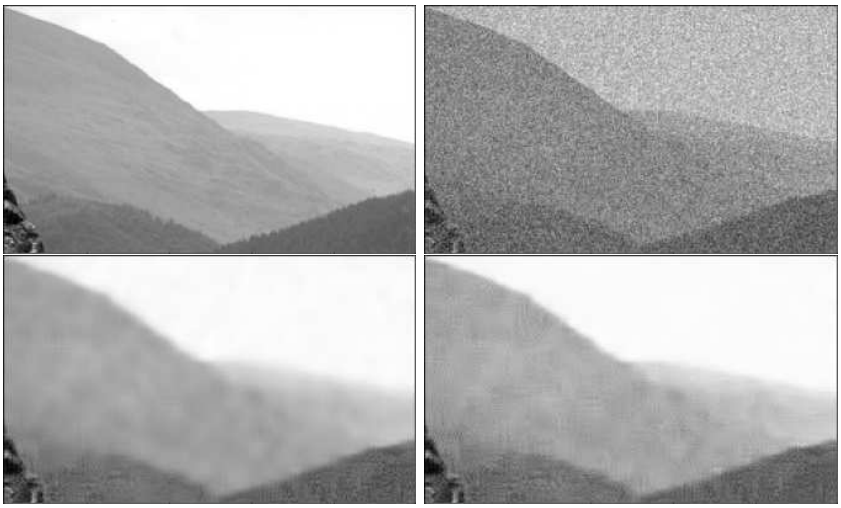

Figure 3.3: Denoiser $\mathcal{D}_{MSE}$ applied to the input image with values $MSE \sim 0.004$, $PSNR \sim 24$: (a) Top left: Original clean image, (b) Top right: Image with added noise, (c) Bottom left: Image denoised using $\mathcal{D}_{MSE}$, (d) Bottom right: Image denoised using $\mathcal{D}^{0.7}_{MIX}$.

Based on this, we can visually assess whether the regularization parameter α, which results in the residual vector closely resembling white noise, offers the optimal balance between denoising effectiveness and preservation of the input image's content. In fact 3.1 compute the largest difference, in absolute value, between the estimated and the theoretical NCP; we can then choose the parameter that minimizes this difference. We finally compared the values of PSNR and SSIM to test quality of the denoised images in terms of noise the first and structure similarity the second. In this way we are able to verify if, using MIX loss function as training function, the algorithm can have an improvement in terms of denoising and perception. Below we show the results plotting the Normalized Cumulated Periodogram of all the cases comparing them with the added white noise; we then plot the values of SSIM and PSNR and visualize them better in a table, together with the deviation $\mathcal{N}$. Finally we show the results displaying the original image, noised and the image denoised with D_{MSE} and $D^{0.7}_{MIX}$, which as we will discuss, improves the task of denoising, preserving structural similarity.

We can take under consideration the images in 3.5. From figure 3.4, or better in the table 3.1, we can observe that the residuals between the original image and the reconstructed one, closest to the white noise, are those related to the MIX loss with $\alpha = 0.7, 0.69$. An ulterior confirm for that is in figure 3.9, which show how the deviation goes with some values of α, taking into account mean and standard deviation for 10 iteration of the test. This is done to assure that the variability of the random choice of the noise does not affect results in a significant way. We computed also mean and standard deviation due to

random settings during different training of the network. We can then compare the values for PSNR and SSIM for different values of α, shown in 3.12 and 3.1, and note that $D^{0.69}_{MIX}$ has the highest value for SSIM, together with $D^{0.7}_{MIX}$. Furthermore, the PSNR values for the first two denoisers are the highest, which means that the best tradeoff is reached by the said values of α. Finally, we can observe that for $\alpha = 0.7$, the deviation 2.45 is minimized. We can show these results in the denoised images in figure 3.5.
We can consider another example in which this is the case, demonstrating that the improvement is usually relevant. We can then take into account image 3.6. In table 3.2 we can observe that, also in this example, the minimal discrepancy of the residual from the straight line is achieved by the denoiser with value of $\alpha = 0.7$, while the highest values of SSIM and PSNR are achieved with $\alpha = 0.6, 0.7$. Nevertheless, comparing the output images 3.6, the best result is achieved by $D^{0.7}_{MIX}$.
We can then consider an example in which the quality values are higher for D_{MSE} respect to $D^{0.7}_{MIX}$, as we can notice in table 3.3. However, in this case the resulting tradeoff is very close for the various denoisers as we can observe in 3.14.
We finally give a last example in which the values for SSIM and PSNR are noticeably higher for $D^{0.6}_{MIX}$ compared to $D^{0.7}_{MIX}$ as shown in 3.4, despite the fact that the residual for this last denoiser approximates better the white noise. We can confirm that in figures 3.15 and 3.4. From these graphs we can also observe that the best tradeoff between SSIM and PSNR is achieved by $D^{0.6}_{MIX}$, and also that the minimal deviation from the straight line is achieved again by $D^{0.7}_{MIX}$. We can have a proof for that in figure 3.8.

In conclusion the best results are achieved for a value of α between 0.7 and 0.6, when the values of PSNR and SSIM are relatively high and the discrepancy from the white noise is minimized. However, the best performance is achieved by $D^{0.7}_{MIX}$, for which the deviation $\mathcal{N}$ is always the minimal one, even though the values of PSNR and SSIM are not the optimal. On the other hand, in almost all the examples, those values are not so far away from the highest ones. This means that may exist a relation between these parameters which plays a role into obtaining a good denoising while preserving the image structures. For instance, if the residual between the denoised image and the noisy original resembles white noise, it suggests that the residual primarily consists of noise information rather than image structures, which could potentially compromise the quality of the reconstructed image.

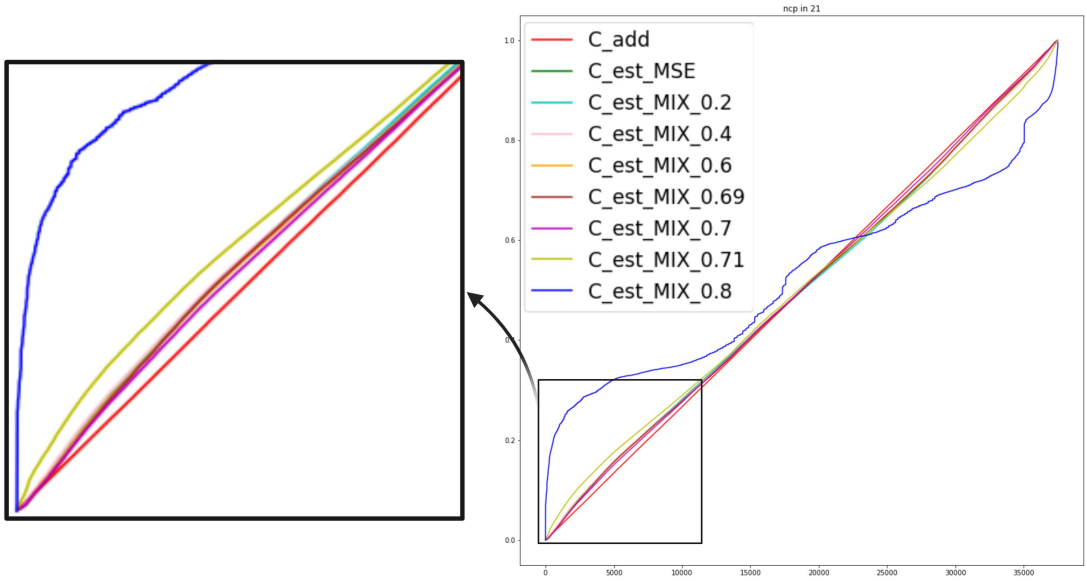

Figure 3.4: Normalized Cumulative Periodogram for the residual related D_{loss} and for the added white noise associated to image 3.5. In the legend we specify which curve is represented by D_{loss}.

	Noise	MSE (MIX_0), std	$MIX_{0.2}$	$MIX_{0.4}$	$MIX_{0.6}$	$MIX_{0.69}$	$MIX_{0.7}$	$MIX_{0.71}$	$MIX_{0.8}$
PSNR_mean	13.537	18.508, 0.1	18.478	18.415	18.640	18.760	18.754	17.170	12.888
PSNR_std	0.03	0.03	0.04	0.03	0.04	0.04	0.04	0.03	0.03
SSIM_mean	0.905	0.948, 0.009	0.934	0.958	0.955	0.964	0.968	0.946	0.901
SSIM_std	0.0004	0.002	0.006	0.0004	0.003	0.003	0.0007	0.001	0.001
$\mathcal{N}$_mean	49.440	433.120, 42.5	474.648	461.568	388.567	386.376	268.484	823.513	3474.57
$\mathcal{N}$_std	20.017	33.8	32.4	31.6	31.1	30.6	28.8	33.9	31.2

Table 3.1: In the table are saved the values of PSNR, SSIM and deviation of the residual $\mathcal{N}$, related to the denoiser D_{loss} for image 3.5 from the straight line; the optimal values are underlined in red. For $\alpha = 0$ we computed mean and standard deviation due to the random settings of noise and dataset during different training processes. Furthermore, for each value of α, we computed mean and standard deviation for 10 iteration of the same image in the testing phase, to check if the random distribution of noise has impact on the results. From the table we can see that it is limited, then the results are reproducible. This considerations can be extended for the other examples.

Figure 3.5: Comparison of implemented denoisers applied to the input image: (a) Top left: Original clean image, (b) Top right: Image with added noise, (c) Bottom left: Image denoised using D_{MSE}, (d) Bottom right: Image denoised using $D^{0.7}_{MIX}$. We can observe that in the image denoised with D_{MSE}, in the monument are lost almost all contents, and the image results to be more smoothed compared to the one denoised with $D^{0.7}_{MIX}$, which preserves details better.

	Noise	MSE (MIX_0), std	$MIX_{0.6}$	$MIX_{0.69}$	$MIX_{0.7}$	$MIX_{0.71}$	$MIX_{0.8}$
PSNR	14.153	20.064, 0.2	20.525	20.383	20.489	18.893	14.611
SSIM	0.912	0.974, 0.01	0.977	0.976	0.977	0.964	0.922
$\mathcal{N}$	45.058	350.909, 49.7	289.684	371.333	208.634	557.801	1186.568

Table 3.2: In the table are saved the values of PSNR, SSIM and deviation of the residual related to the denoiser D_{loss} for image 3.6 from the white noise; the optimal values are underlined in red.

	Noise	MSE (MIX_0), std	$MIX_{0.6}$	$MIX_{0.69}$	$MIX_{0.7}$	$MIX_{0.71}$	$MIX_{0.8}$
PSNR	14.016	22.783, 0.2	23.116	23.146	22.685	22.807	13.722
SSIM	0.886	0.983, 0.001	0.984	0.984	0.983	0.982	0.885
$\mathcal{N}$	45.058	258.002, 15.5	243.565	311.308	192.680	295.051	2492.851

Table 3.3: In the table are saved the values of PSNR, SSIM and deviation of the residual related to the denoiser D_{loss} for image 3.7 from the white noise; the optimal values are underlined in red. For a better comparison between D_{MSE} and $D^{0.7}_{MIX}$ the PSNR values, which are higher for MSE, are underlined in blue.

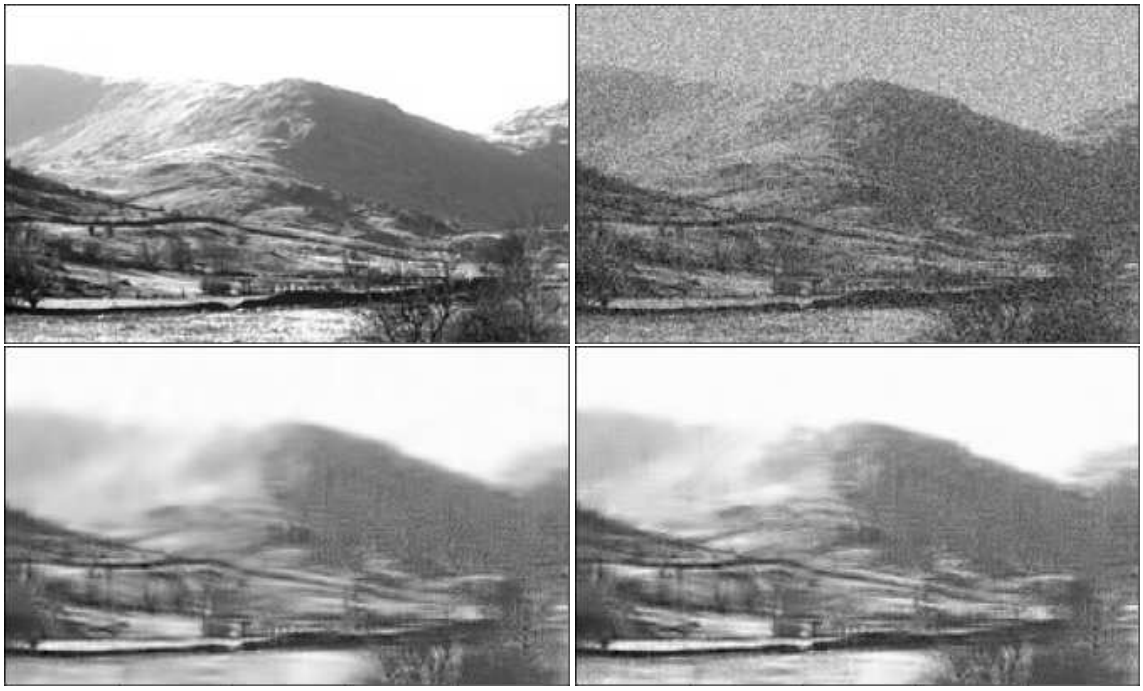

Figure 3.6: Comparison of implemented denoisers applied to the input image: (a) Top left: Original clean image, (b) Top right: Image with added noise, (c) Bottom left: Image denoised using D_{MSE}, (d) Bottom right: Image denoised using $D^{0.7}_{MIX}$. Here image (c), the one denoised with D_{MSE}, results smooth in the brightest profile of the mountain, losing all the shades in that part. Although, image (d) results more defined.

	Noise	MSE (MIX_0), std	$MIX_{0.6}$	$MIX_{0.69}$	$MIX_{0.7}$	$MIX_{0.71}$	$MIX_{0.8}$
PSNR	13.876	21.879, 0.3	22.297	22.423	21.962	21.722	13.487
SSIM	0.880	0.977, 0.002	0.980	0.980	0.979	0.972	0.877
$\mathcal{N}$	45.058	209.316, 12.7	191.667	232.977	120.121	264.742	2408.696

Table 3.4: In the table are saved the values of PSNR, SSIM and deviation of the residual related to the denoiser D_{loss} for image 3.8 from the white noise; the optimal values are underlined in red. For a better comparison between D_{MSE} and $D^{0.7}_{MIX}$ the PSNR values, are underlined in blue.

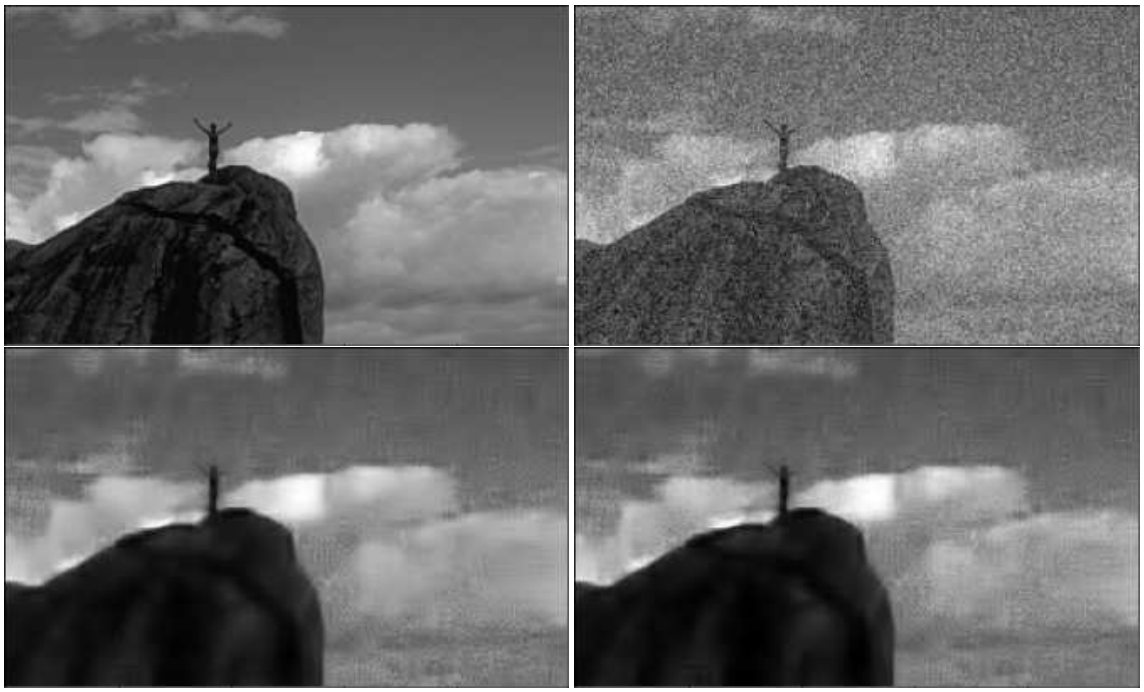

Figure 3.7: Comparison of implemented denoisers applied to the input image: (a) Top left: Original clean image, (b) Top right: Image with added noise, (c) Bottom left: Image denoised using D_{MSE}, (d) Bottom right: Image denoised using $D^{0.7}_{MIX}$. In this case we can observe that the improvement is less noticeable compared to the previous figures. The image (d) appear most degraded for example in the darkest clouds. However, image (c) appears smoother, losing some shades variations. Also the arms of the person are more neat in image (d).

Figure 3.8: Comparison of implemented denoisers applied to the input image: (a) Top left: Original clean image, (b) Top right: Image with added noise, (c) Bottom left: Image denoised using D_{MSE}, (d) Bottom right: Image denoised using $D^{0.7}_{MIX}$, (e) Bottom right: Image denoised using $D^{0.6}_{MIX}$, (f) Bottom right: Image denoised using $D^{0.69}_{MIX}$. In this figure we added also the outputs for $D^{0.6}_{MIX}$ and $D^{0.69}_{MIX}$, to visually show that, although the highest values of SSIM and PSNR of these compared to $D^{0.7}_{MIX}$, image (d) results less smooth and presents some splotchy artifacts. However, it preserves better structures, for examples, the different shades in the clouds.

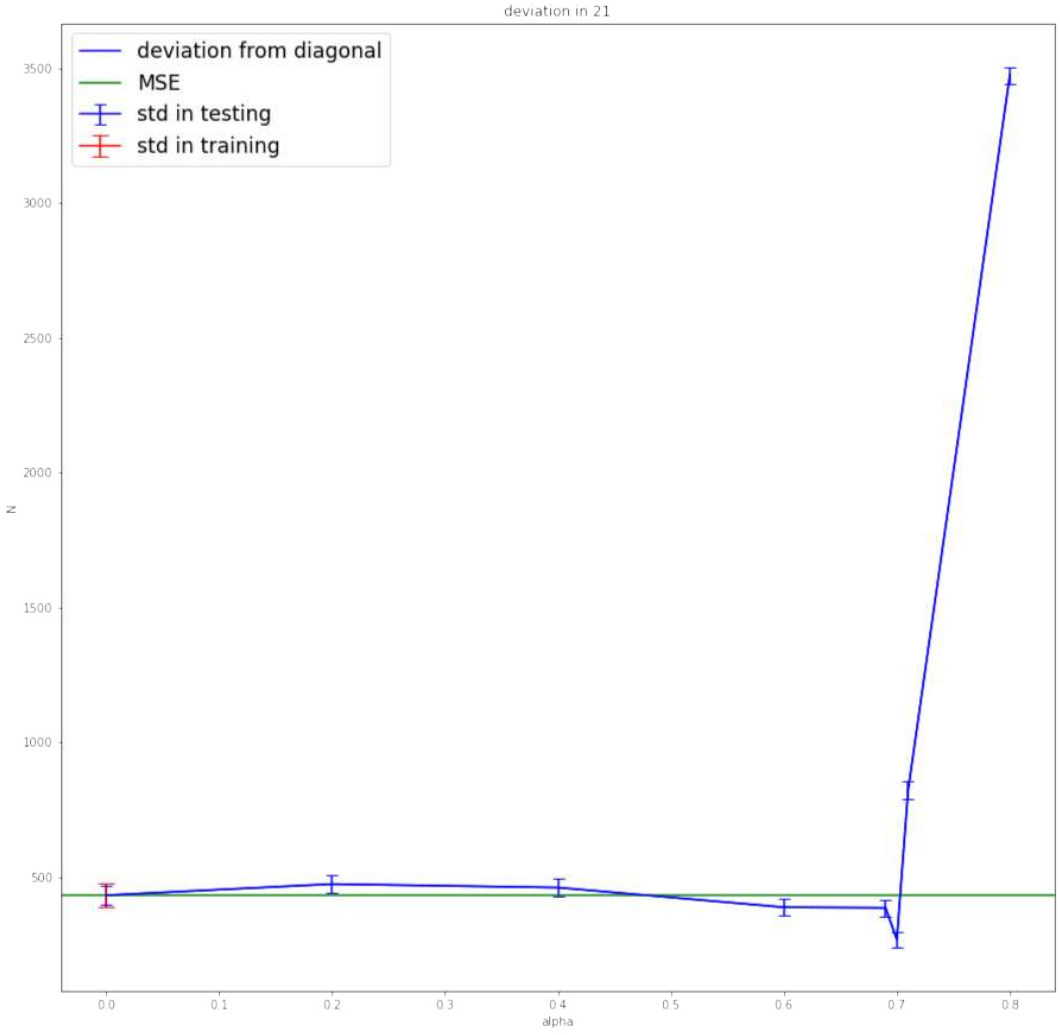

Figure 3.9: Plot of deviation $\mathcal{N}$ 3.1 with respect to value α in comparison with the deviation with respect to $\alpha = 0$ (MSE), related to figure 3.5. The graph takes into account mean and standard deviation during 10 test iterations and different training. See table 3.1 for specific values. The plot trend can be extended to the other examples too.

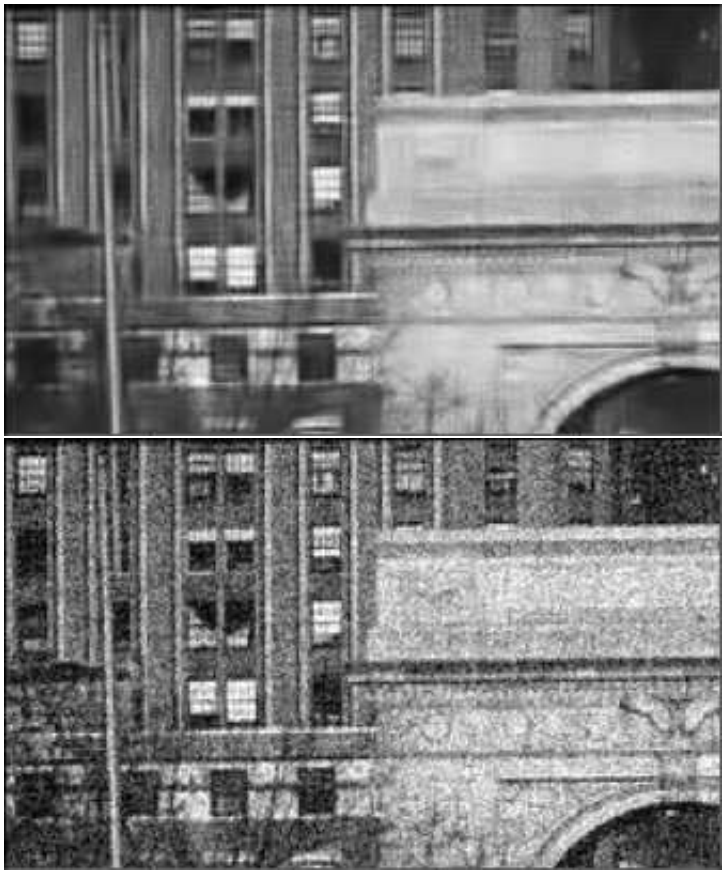

Figure 3.10: The above image shows image in 3.5 denoised using $\mathcal{D}^{0.7}_{MIX}$, while in the bottom is shown the image denoised using $\mathcal{D}^{0.8}_{MIX}$. Observing figure 3.9 too, we can confirm that the second image still presents noise.

Figure 3.11: The first image shows image in 3.5 denoised using $\mathcal{D}^{0.7}_{MIX}$, while the second shows the image denoised using $\mathcal{D}^{0.8}_{MIX}$. With the acknowledge that with $\alpha = 0.7$ we reached the optimal deviation from white noise, while with $\alpha = 0.8$ this value explodes, we can confirm that the second image, still presents noise.

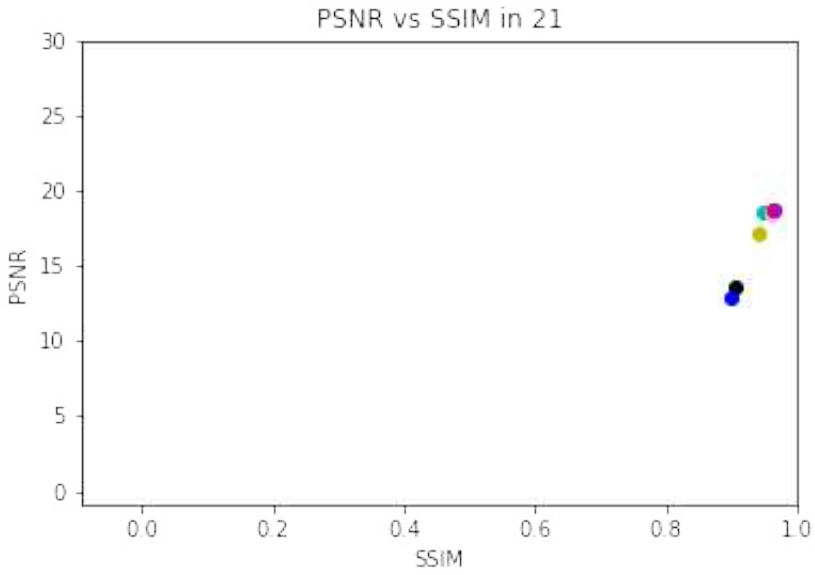

Figure 3.12: Plot of Structure Similarity Index Measure (SSIM) vs Peak Signal to Noise Ratio (PSNR) for image 3.5. The optimal result maximizing PSNR together with SSIM is given by $D^{0.7}_{MIX}$. For a legend specifying the values see the legend in 3.4.

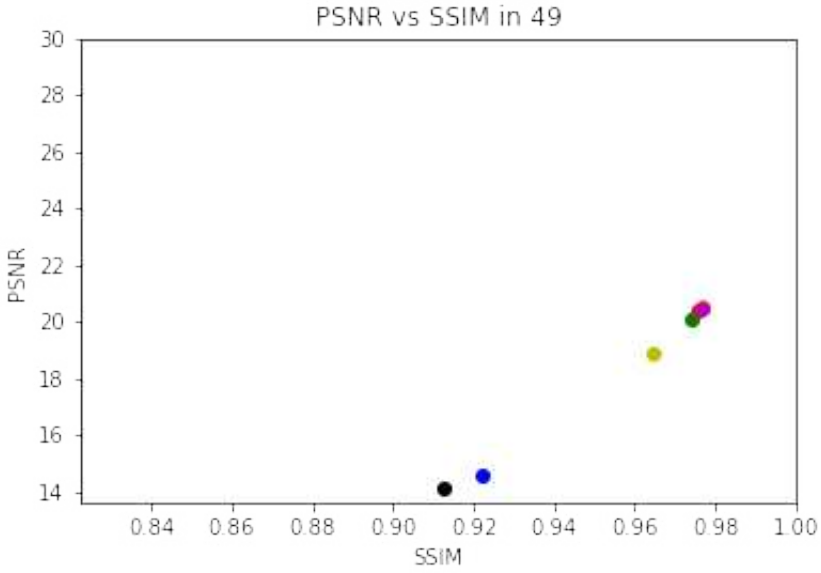

Figure 3.13: Plot of Structure Similarity Index Measure (SSIM) vs Peak Signal to Noise Ratio (PSNR) for image 3.6. The highest values for both PSNR and SSIM are achieved by $D^{0.6}_{MIX}$ followed by $D^{0.7}_{MIX}$. See legend in figure 3.4.

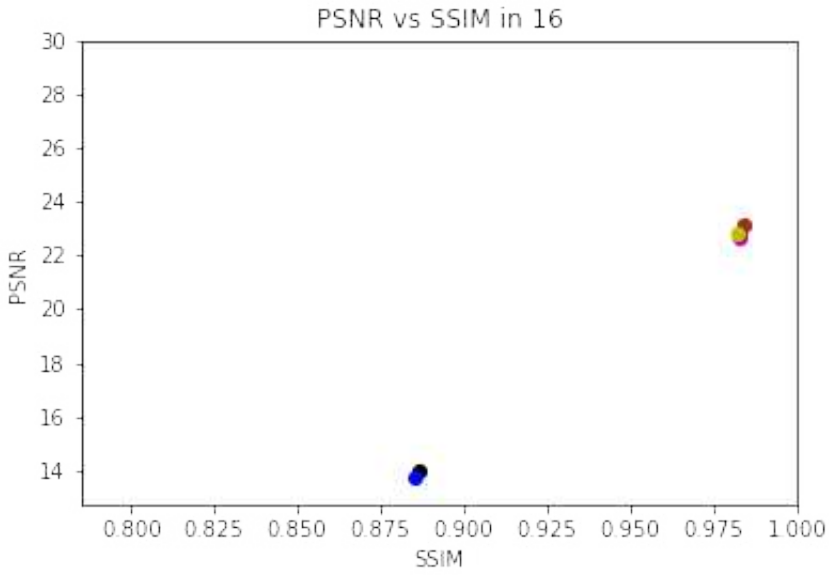

Figure 3.14: Plot of Structure Similarity Index Measure (SSIM) vs Peak Signal to Noise Ratio (PSNR) for image 3.7. Notice that the values of SSIM are very close for the denoisers we are considering. See legend in figure 3.4 to specify the values.

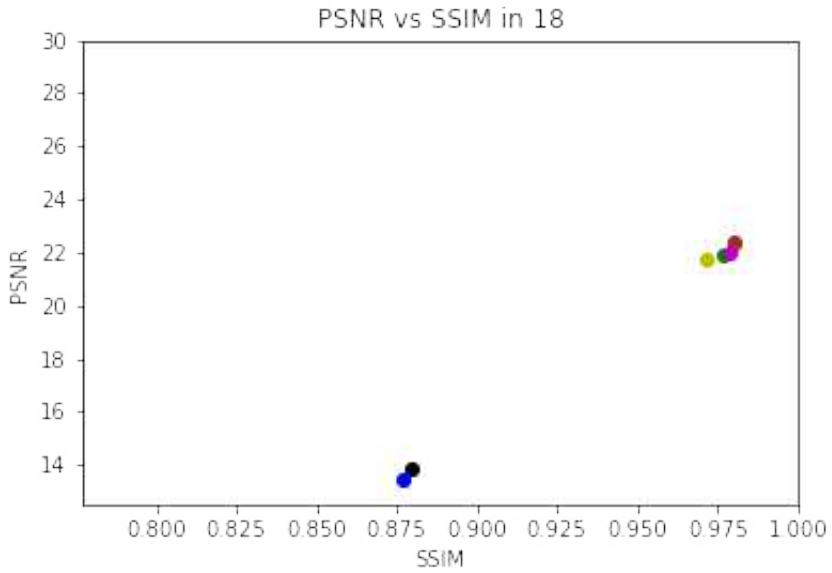

Figure 3.15: Plot of Structure Similarity Index Measure (SSIM) vs Peak Signal to Noise Ratio (PSNR) for image 3.8. Here the optimal values are achieved by $D^{0.69}_{MIX}$ and $D^{0.6}_{MIX}$. See legend in figure 3.4.

Conclusions

In this thesis, we have studied the improvement of the K-SVD algorithm using deep learning, applied on the denoising of images. As mentioned in the previous chapters, we considered an algorithm studied and implemented few years ago and analyzed its strengths and weaknesses, starting from the baseline. Taking into considerations studies related to the problem of denoising while preserving the image content, we acknowledged the existence of a perception-distortion tradeoff. This helped us to focus not only on the enhancement of the denoising task during the training phase of the network, but also on minimizing the loss of structure similarity between the original image and the denoised one. As a first step we focused on the most commonly used loss function in the literature: the Mean Squared Error loss function, to determine if it is effectively the optimal choice for training the network, or if there could be improvements. By comparing the qualitative performance of the algorithm, indicated by a low MSE value, with the image outputs, we were able to observe that while the images were correctly denoised, they appeared smooth and exhibited a loss of image details.

Furthermore, inspired by recent studies which combine different loss functions, we choose to preserve the MSE which is suitable for denoising high frequencies images, combining it with a function that compute the structure similarity, luminance and contrast between images - the Structural Similarity Index Measure. However, an important step was to decide how to weight the two components of the function, to achieve the best tradeoff in terms of noise removal and preserving image contents. For this, we were inspired by a study that provides a criterion for choosing regularization parameters by exploiting residual information for discrete ill-posed problems. This criterion involves computing the Normalized Cumulative Periodogram of the residual and noise, where white noise should exhibit a flat power spectrum and lie within a straight line. Specifically the criterion addresses the choice of the regularization parameter on the computed residual that minimize the maximal deviation from the straight line.

We applied this study in a novel context, focused on the choice of the parameter to guarantee the best perception-distortion tradeoff. Using this criterion we considered the residual between the denoised image and the one to which has been added noise, and it is then compared to the original added noise. Analyzing the behaviour of the residual for various trained denoisers, depending on

the regularization parameter α, we found that a specific value, whose residual optimize the deviation from the straight line, assumes good values in terms of quality measured with Peak to Signal Noise Ratio, which depends on the value of Mean Squared Error, and the Structural Similarity Index Measure. Most importantly, visually, it results in the best reconstructed image, enhancing our initial results. This is attributed to the fact that the residual, being dominated on noise components, contains as less remaining image contents as possible, resulting in a reconstructed image more similar to the original one.

In conclusion, further research could explore this new approach to test image quality in denoising tasks, improving not only the performance of the studied algorithm, for example exploring more values of α, but also other architectures related to image processing, while maintaining control over how the network operates.

Implemented deep K-SVD algorithm

Listing 1: Noise Addiction

```
def __getitem__(self, idx):
    idx_im, idx_sub_image =  np.unravel_index(idx, (number_images,
    ↪  number_sub_images))

    image = self.dataset_list[idx_im]
    sub_image = self.extract_sub_image_from_image(image,
    ↪  self.sub_image_size, idx_sub_image)

    np.random.seed(idx)
    noise = np.random.randn(self.sub_image_size,
    ↪  self.sub_image_size)

    sub_image_noise = sub_image + self.sigma * noise
```

Listing 2: Patches Selection

```
patches = np.array([in_out_patch.R(Y, patch_size, k) for k in
↪  range(number_patches)])
np.save('patches_%d.npy', patches)
```

where `in_out_patch.R(Y, patch_size, k)` is implemented as follows

```
def R(image, patch_size, k):
    """given an image of size=image_size select from it the kth
       patch of size p=patch_size, and reshape it to a vector"""

    N, C, w, h = image.shape

    if k < 0 or k >= (w - patch_size + 1) * (h - patch_size + 1):
```

```
        raise ValueError('Index out of range')

    row_start = int(k // (h - patch_size + 1))
    col_start = int(k % (h - patch_size + 1))
    patch = image[row_start : row_start + patch_size,
            col_start : col_start + patch_size]

    return np.reshape(patch, (patch_size**2,1))
```

Listing 3: MLP Lambda Evaluation

```
def MLP(b0, b1, b2, b3, A0, A1, A2, A3, yks, activation):
    argnu = A0 @ yks + b0
    nu = argnu.clip(min=0)#ReLU
    argmu = A1 @ nu + b1
    mu = argmu.clip(min=0)#ReLU
    argeta = A2 @ mu + b2
    eta = argeta.clip(min=0)
    arglam = A3 @ eta + b3


    if activation == 'ReLU':
        lam = arglam.clip(min=0)

    if activation == 'Identity':
        lam = arglam

    #save output MLP
    keys =
    ↪  ['argnu','nu','argmu','mu','argeta','eta','arglam','lambda']
    values = [argnu, nu, argmu, mu, argeta, eta, arglam, lam]
    MLP_collection = {k : v for k, v in zip(keys, values)}

    return MLP_collection
```

Listing 4: Sparse Coding

```
self.MLP = MLP.MLP(self.b0, self.b1, self.b2, self.b3, self.A0,
↪  self.A1, self.A2, self.A3, patches, self.activation)
lam = self.MLP['lambda']

l = lam / c
y = Dt @ patches

z = soft_thresh(y, l)
z_list[i][0] = z
```

```
for t in range(T):
    v = S @ z + (1 / c) * y
    dv_S, dl_S = Partial_Sk(v, l)
    z = soft_thresh(v, l)
    dv_S_all[t] = dv_S
    dl_S_all[t] = dl_S
    z_list[t+1] = z
```

Listing 5: Patch Reconstruction

```
image_size = Y.shape
x_pred = Dict @ z
x_pred = w * x_pred

num = in_out_patch.num(x_pred, image_size, number_patches)
den = in_out_patch.den(w, image_size, number_patches)

res = num / den
```

with,

```
def den(weight, size, number_patches):
    """compute the weighted normalization of the reconstructed
    ↪  image"""
    Rkt_w = lambda k: Rt(weight, size, k)
    den = np.sum(np.array([Rkt_w(k) for k in
    ↪  range(number_patches)]), axis=0)
    return den

def num(sparse, size, number_patches):
    """compute the weighted reconstructed image"""
    Rkt_s = lambda k: Rt(sparse[k], size, k)
    num = np.sum(np.array([Rkt_s(k) for k in
    ↪  range(number_patches)]), axis=0)
    return num
```

Listing 6: Weights Initialization

```
def initialize_weights(patch_size, m, D_in, H_1, H_2, H_3,
↪  D_out_lam):

    bound0=np.sqrt(1/D_in)
    bound1=np.sqrt(1/H_1)
    bound2=np.sqrt(1/H_2)
    bound3=np.sqrt(1/H_3)
```

```
7
8       ## initialize Dict using Discrete Cosinus Transform
9       Dict_init = init_dct(patch_size, m)
10      ## initialize c as the squared norm of Dict
11      c_init = np.reshape(np.float64(np.array((linalg.norm(Dict_init,
        ↪  ord=2))** 2)), (1,1))
12      ## initialize w using normal distribution
13      w_init = np.float64(np.random.normal(loc=1, scale=1 / 10 *
        ↪  np.ones((patch_size ** 2,1))))
14
15      ## initialize the MLP weigths using uniform distribution
16      A0_init = np.random.uniform(-bound0, bound0, size=(H_1, D_in))
17      A1_init = np.random.uniform(-bound1, bound1, size=(H_2, H_1))
18      A2_init = np.random.uniform(-bound2, bound2, size=(H_3, H_2))
19      A3_init = np.random.uniform(-bound3, bound3, size=(D_out_lam,
        ↪  H_3))
20      b0_init = np.random.uniform(-bound0, bound0, size=(H_1, 1))
21      b1_init = np.random.uniform(-bound1, bound1, size=(H_2, 1))
22      b2_init = np.random.uniform(-bound2, bound2, size=(H_3, 1))
23      b3_init = np.random.uniform(-bound3, bound3, size=(D_out_lam,
        ↪  1))
24
25
26      initialize_weights = {'Dict': Dict_init, 'c': c_init, 'w':
        ↪  w_init, 'b0': b0_init, 'b1': b1_init, 'b2': b2_init, 'b3':
        ↪  b3_init, 'A0': A0_init, 'A1': A1_init, 'A2': A2_init, 'A3':
        ↪  A3_init}
27
28      return initialize_weights
```

Bibliography

[1] M. Scetbon, M. Elad, and P. Milanfar, *Deep K-SVD denoising*, IEEE Transactions on Image Processing, (2021), vol. 30, pp. 5944-5955.

[2] M. Aharon, M. Elad, and A. Bruckstein, *K-SVD: An algorithm for designing overcomplete dictionaries for sparse representation*, IEEE Transactions on Signal Processing, (Nov. 2006), vol. 54, no. 11, pp. 4311–4322.

[3] *https://github.com/meyerscetbon/Deep-K-SVD*,

[4] Z. Wang, A.C. Bovik, H.R. Sheikh, E.P. Simoncelli: *Image quality assessment: from error visibility to structural similarity*, IEEE Transactions on Image Processing, (2004), vol. 13, no.4, pp. 600–612.

[5] H. Zhao, O. Gallo, I. Frosio, and J. Kautz, *Loss Functions for Image Restoration With Neural Networks*, IEEE Transactions on Computational Imaging, (Jan 2017), vol. 3, no. 1.

[6] H. Pishro-Nik, *Introduction to probability, statistics, and random processes*, available at https://www.probabilitycourse.com, Kappa Research LLC, 2014.

[7] M. Elad, B. Kawar, G. Vaksman, *Image Denoising: The Deep Learning Revolution and Beyond - A Survey Paper*, SIAM Journal on Imaging Sciences, (2023), vol. 16, no. 3, pp. 1594–1654.

[8] Y. Blau, T. Michaeli, *The Perception-Distortion Tradeoff*, Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition, (2018), pp. 6228–6237, 8578750.

[9] Y. Yang, C.-H. Chou, J.P. Allebach, *Mix-loss trained bias-removed blind image denoising network*, IS and T International Symposium on Electronic Imaging Science and Technology, (2022), vol. 34, no. 8, 288.

[10] D. L. Donoho, M. Elad, and V. N. Temlyakov, *Stable Recovery of Sparse Overcomplete Representations in the Presence of Noise*, IEEE Transactions on Inf. Theory, (Jan. 2006), vol. 52, no. 1, pp. 6–18.

[11] D. L. Donoho and M. Elad, *Optimally sparse representation in general (nonorthogonal) dictionaries via L1 minimization*, Proc. Nat. Acad. Sci. USA, (Mar. 2003), vol. 100, no. 5, pp. 2197–2202.

[12] I. Daubechies, M. Defrise, and C. D. Mol, *An iterative thresholding algorithm for linear inverse problems with a sparsity constraint*, Commun. Pure Appl. Math., (2004), vol. 57, no. 11, pp. 1413–1457.

[13] K. Gregor and Y. LeCun, *Learning Fast Approximations of Sparse Coding*, in Proc. ICML. Norristown, PA, USA: Omnipress, (2010), pp. 399–406.

[14] P.C. Hansen, M.E. Kilmer, R.H. Kjeldsen, *Exploiting residual information in the parameter choice for discrete ill-posed problems*, BIT Numerical Mathematics, (2006), vol.46, no.1, pp. 41–59.

[15] W. A. Fuller, *Introduction to Statistical Time Series*, 2nd edn., Wiley, New York, 1996.

[16] A. Horé, D. Ziou, *Image quality metrics: PSNR vs. SSIM* , Proceedings - International Conference on Pattern Recognition, (2010), no. 5596999, pp. 2366-2369.