POLITECNICO DI TORINO

Master's degree
in Data Science and Engineering

Tesi di Laurea

Patch-based learning of space-variant hyperparameters in variational image restoration

Relatori
prof. Salvatore Ivan Trapasso
prof. Luca Calatroni
prof. Xavier Descombes

Candidato
Claudio Fantasia

Anno Accademico 2023-2024

Summary

Solving an inverse imaging problem consists in the task of reconstructing an unknown image from observed, often incomplete noisy data. A standard example describing a linear degradation process is the problem of image deconvolution where the operator describing the blurring process (forward operator) is a convolution matrix.
Inverse problems are typically ill-posed, that is small perturbations in the observed image (input) lead to significant perturbations in the desired image (output). A standard paradigm overcoming such instabilities consists in considering a regularised formulation of the problem in order to guarantee stable reconstruction of the solution. Regularisation is combined with a data-fitting term along with hyperparameters balancing the two. Total Variation regularisation is a popular regulariser in the context of imaging due to its enhanced edge-preserving behaviour and noise smoothing properties in comparison to other, simpler, regularisations (e.g. Tikhonov, l_1 norm).
From a Bayesian perspective, Total Variation (TV) regularisation implicitly assumes a space-invariant, one-parameter half-Laplacian distribution (hLd) for the gradient magnitudes of the true image. However, this assumption is often restrictive in the modelling of natural images, whose statistics are often characterised by heterogeneous and highly oscillating behaviour. Ideally, in smooth regions, stronger regularization is desired, while in textured areas, less regularisation is preferred.
This naturally led us to a new formulation where, for each pixel, an adaptive regularisation parameter weights the local image contents. This model better captures the space-variant (SV) nature of gradient distribution throughout the image. Numerically, such space variant model can be efficiently solved by means of accelerated first-order algorithms such as, e.g., the Fast Iterative Thresholding Algorithm (FISTA) endowed with a non-monotone backtracking strategy to select optimal step-sizes.
To compute effectively these weights, differently from the scalar case, where in most cases a brute-force approach (grid search) is used to select a optimal parameter maximising suitable quality metrics (e.g., PSNR, SSIM), in the SV approach such strategy is not feasible due to the potentially large size of the image considered. To mitigate this problem, we consider a deep learning-based patch-based approach where, for each image patch (of fixed size) a golden section algorithm is used to select an optimal scalar parameter by suitable comparison with ground truth data. The value computed can be then used as an optimal value in the centre of the selected patch and the map of space-variant parameters can be built in a sliding window fashion.
By using natural image datasets of different size and noise distribution/intensity, we then implemented a supervised learning approach where a light neural network is trained to predict the optimal parameter in a pixel by looking only at a small patch around the pixel itself, leading to fast and effective inference after suitable training.

The space-variant reconstruction approach obtained better results in terms of SSIM and PSNR with respect to the global reconstruction leading to better detail preservation and reduced over-smoothing in textured regions. Also, our ML approach drastically decreased the computational time required to compute the parameters map without the use of ground truth, an essential feature in real-world applications.

Contents

List of Tables 8

List of Figures 9

1 Introduction to inverse problems 13
- 1.1 Inverse problems . . . 13
- 1.2 Bayesian interpretation . . . 14
- 1.3 Data fidelity . . . 15
  - 1.3.1 Gaussian data fidelity . . . 15
  - 1.3.2 Poisson data fidelity . . . 16
- 1.4 Regularisation . . . 17
- 1.5 Optimization . . . 19
  - 1.5.1 Gradient Descent . . . 19
  - 1.5.2 Accelerated gradient descent . . . 21
  - 1.5.3 Fast iterative shrinkage-thresholding algorithm (FISTA) . . . 21
- 1.6 Image processing metrics . . . 22

2 Hyperparameters estimation 23
- 2.1 Space invariant regularisation in the Gaussian case . . . 24
  - 2.1.1 Algorithmic settings . . . 26
- 2.2 Scalar regularisation in the Poisson case . . . 28
  - 2.2.1 Algorthmic settings . . . 30
- 2.3 Space-variant regularisation . . . 32

3 Patch-based learning 35
  - 3.0.1 Notation . . . 35
- 3.1 Berkley dataset . . . 36
- 3.2 Training dataset . . . 37
- 3.3 Deep parameter estimation . . . 40
- 3.4 Architecture . . . 41
- 3.5 Training . . . 41
- 3.6 Inference time . . . 43

4 Results 45
- 4.1 Classification task . . . 46
- 4.2 Gaussian denoising . . . 48
  - 4.2.1 Low noise ($\sigma^2_{AWGN} = 0.01$) . . . 48
  - 4.2.2 Medium noise ($\sigma^2_{AWGN} = 0.02$) . . . 50

4.2.3 Enhancing robustness 51
4.2.4 Ablation study on patch sizes 52
4.3 Gaussian deblurring 54
4.4 Poisson denoising 56

5 Conclusions and outlook 59

List of Tables

3.1 Notation 35
3.2 Summary statistics 39
3.3 Details of the first architecture. 42
3.4 Details of the second architecture. 42
4.1 Summary of results for six datasets, including R^2 values, tested noise levels, and SSIM improvement metrics. 45

List of Figures

1.1 Solution of exact square solution with only blur. 17
1.2 Solution of exact square solution with blur and small noise added. 18
1.3 Transition from clean image to corrupted. 19
1.4 Solution of iterative methods with only L2 Fidelity term. 19
1.5 Different effects on the solution based on the regulariser used. 19
1.6 Difference between GD and AGD directions. 21
2.1 Ground truth image and its noisy version. Noise is Gaussian with $\sigma^2_{AWGN} = 0.1$ and Gaussian blur with $\sigma^2_{PSF} = 1.5$. 23
2.2 Different reconstruction obtained given by the choice of μ. 24
2.3 Global and local image content prior probability. 25
2.4 Brute force approach for computing the optimal parameter. 26
2.5 Dot plot of the value examined by golden-section algorithm. 26
2.6 Different rate of convergence between GD and FISTA. 27
2.7 Solution obtained by the solvers after 160 iterations. 28
2.8 Different image corruption based on average pixel's intensity. 29
2.9 Step sizes of FISTA along the iterations. 32
2.10 Blurry and noisy image to reconstruct. 34
2.11 Reconstructed image with its patch-wise parameters map. 34
3.1 Some examples from Berkley dataset. 36
3.2 Histograms of labels between the dataset with $\sigma^2 = 0.01$ and the one with $\sigma^2 = 0.02$. 38
3.3 Histogram of the dataset labels affected by blur: $\boldsymbol{\mu}^{\mathrm{GT}_4}$. 39
3.4 Histogram of the dataset labels with reduced patch size $\boldsymbol{\mu}^{\mathrm{GT}_5}$. 39
3.5 Histograms of the labels of the Poisson noise dataset $\boldsymbol{\mu}^{\mathrm{GT}_6}$. 40
3.6 CNN architecture for the estimation of the sub-optimal parameter. 41
4.1 Validation loss for classification task. 46
4.2 Classification of Gaussian noise based on different intensities. 47
4.3 Classification of Poisson noise based on different intensities. 47
4.4 Losses for denoising task with $\sigma^2_{AWGN} = 0.01$. 48
4.5 Noisy image with its scalar and space-variant reconstruction for $\sigma^2_{AWGN} = 0.01$ 48
4.6 3D graph-2D colorbar of the parameters map for $\sigma^2_{AWGN} = 0.01$ Gaussian denoising. 49
4.7 Images with same SSIM values, but different preserved texture. 49
4.8 Losses for denoising task with $\sigma^2_{AWGN} = 0.02$. 50
4.9 Noisy image with its scalar and space-variant reconstruction for $\sigma^2_{AWGN} = 0.02$. 50
4.10 2D colorbar for the Gaussian denoising tested on $\sigma^2_{AWGN} = 0.01, \sigma^2_{AWGN} = 0.02$. 51
4.11 Losses for denoising task with $\sigma^2_{AWGN} = 0.01$,0.02. 51

4.12 Left: noisy images with different Gaussian noise levels. Right: scalar and space variant reconstructions for $\sigma^2_{AWGN} = 0.015$ and $\sigma^2_{AWGN} = 0.02$ using the architecture trained on mixed noises. 52
4.13 Space-variant reconstruction by architectures trained on 64×64 and 32×32. . 53
4.14 Better reconstruction of fine-grained details in the denoising case with $\sigma^2_{AWGN} = 0.01$. 54
4.15 Losses for deblurring task with $\sigma^2_{AWGN} = 0.01, \sigma^2_{PSF} = 1.0$. 55
4.16 Blurred image with its global and space-variant reconstruction for $\sigma^2_{AWGN} = 0.01, \sigma^2_{PSF} = 1.0$. 55
4.17 Losses(big architecture) for denoising task in Poisson case for $\alpha_1 = 45, \alpha_2 = 15, \alpha_3 = 5$. 56
4.18 Losses(small architecture) for denoising task in Poisson case for $\alpha_1 = 45, \alpha_2 = 15, \alpha_3 = 5$. 56
4.19 (Poisson) noisy image with its global and space-variant reconstruction for $\alpha = 45$. 57
4.20 (Poisson) noisy image with its global and space-variant reconstruction for $\alpha = 15$. 58

Bibliography

Babak Maboudi Afkham, Julianne Chung, and Matthias Chung. Learning regularization parameters of inverse problems via deep neural networks. *Inverse Problems*, 37(10):105017, 2021.

Amir Beck and Marc Teboulle. A fast iterative shrinkage-thresholding algorithm for linear inverse problems. *SIAM journal on imaging sciences*, 2(1):183–202, 2009.

Luca Calatroni and Antonin Chambolle. Backtracking strategies for accelerated descent methods with smooth composite objectives. *SIAM journal on optimization*, 29(3):1772–1798, 2019.

Luca Calatroni, Cao Chung, Juan Carlos De Los Reyes, Carola-Bibiane Schönlieb, and Tuomo Valkonen. Bilevel approaches for learning of variational imaging models. *Variational Methods: In Imaging and Geometric Control*, 18(252):2, 2017.

Luca Calatroni, Alessandro Lanza, Monica Pragliola, and Fiorella Sgallari. A flexible space-variant anisotropic regularization for image restoration with automated parameter selection. *SIAM Journal on Imaging Sciences*, 12(2):1001–1037, 2019.

Luca Calatroni, Alessandro Lanza, Monica Pragliola, and Fiorella Sgallari. Adaptive parameter selection for weighted-tv image reconstruction problems. In *Journal of Physics: Conference Series*, volume 1476, page 012003. IOP Publishing, 2020.

Antonin Chambolle and Thomas Pock. A first-order primal-dual algorithm for convex problems with applications to imaging. *Journal of mathematical imaging and vision*, 40:120–145, 2011.

G.Peyre. nt_toolbox - GitHub. URL https://github.com/gpeyre/numerical-tours/tree/master/python/nt_toolbox. Numerical tours GitHub repository.

Zachary T Harmany, Roummel F Marcia, and Rebecca M Willett. This is spiral-tap: Sparse poisson intensity reconstruction algorithms—theory and practice. *IEEE Transactions on Image Processing*, 21(3):1084–1096, 2011.

Rim Rekik Dit Nekhili, Xavier Descombes, and Luca Calatroni. A hybrid approach combining cnns and variational modelling for blind image denoising. 2022.

Yurii Nesterov. A method for solving the convex programming problem with convergence rate o (1/k2). In *Dokl akad nauk Sssr*, volume 269, page 543, 1983.

Monica Pragliola, Luca Calatroni, Alessandro Lanza, and Fiorella Sgallari. On and beyond total variation regularization in imaging: the role of space variance. *SIAM Review*, 65(3): 601–685, 2023.

Leonid I. Rudin, Stanley Osher, and Emad Fatemi. Nonlinear total variation based noise removal algorithms. *Physica D: Nonlinear Phenomena*, 60(1):259–268, 1992. ISSN 0167-2789.

Chapter 1

Introduction to inverse problems

In this chapter, a brief introduction about imaging inverse problems will be provided along with a quick review on how to solve them. We will first present these problems from a statistical perspective and then move to an optimization point of view. We will discuss the various regularization techniques that can be applied to ensure stable procedures to deal with the intrinsic ill-posedness of the problems considered, as well as some optimization methods and algorithms that can address it from a numerical viewpoint. In conclusion, we will introduce some quality metrics to evaluate the effectiveness of the reconstruction procedure.

1.1 Inverse problems

The objective of an inverse problem is to retrieve a quantity $\boldsymbol{x}$ having access to the observed data $\boldsymbol{y}$, which is know to be the results of a measurement process that is defined by an (possibly known) operator $\boldsymbol{A}$. This process describes how the observed data $\boldsymbol{y}$ are derived from the system under investigation and is often referred to as the forward model.
Mathematically, the forward model maps the unknown quantity $\boldsymbol{x}$, which we want to estimate, into observable quantities. This mapping is represented by a forward operator $\boldsymbol{A} : \mathbb{R}^n \to \mathbb{R}^n$. The inverse problem can then be expressed in the following form:

$$\boldsymbol{A}\boldsymbol{x} = \boldsymbol{y}, \tag{1.1}$$

where $\boldsymbol{A} \in \mathbb{R}^{n \times n}$ describe the relationship between the unknown $\boldsymbol{x}$ and the observations $\boldsymbol{y}$.
In real-world scenarios, measurements are always corrupted by some level of noise. This noise can come from various sources such as sensor inaccuracies, external disturbances, or errors in the measurement process itself. The noise is often modeled as a sampling process from some probability distribution, where the most common one is the Gaussian distribution.
In real applications, inverse problems are typically ill-posed, meaning that they do not satisfy the conditions for being well-posed. The definition of well-posed problem is the following:

Definition 1.1 (Well-posed inverse problem) *A problem is said to be well-posed if the following hold:*

- *The solution of the problem exists*

- *The solution is unique*
- *The solution has a continue dependence on the data (stability)*

If one of the latters does not hold the problem is defined as ill-posed.

1.2 Bayesian interpretation

A standard paradigm for the solution of ill-posed inverse problems consists in the formulation of a minimization problem composed of a data fidelity term $\mathcal{D}$ and a regularisation term $\mathcal{R}$ encoding a priori assumptions on the desired solution (sparsity, smoothness) in the form

$$\min_{\boldsymbol{x}\in\mathbb{R}^n} \mathcal{D}(\boldsymbol{A}\boldsymbol{x}, \boldsymbol{y}) + \mathcal{R}(\boldsymbol{x}; \boldsymbol{\theta}). \tag{1.2}$$

To balance the effect of these two terms, a vector of hyperparameters $\boldsymbol{\theta} \in \mathbb{R}^d$ appears. From a statistical viewpoint, those parameters characterise the corresponding image prior distributions. The estimation of these parameters is necessary for obtaining good reconstruction quality.
A prominent subfield of inverse problems is that of imaging, with applications in chemistry, biology, geosciences, and medicine. Various methods for imaging have been developed over the previous four decades and have been partly made available for practical use.
Firstly, we tackle these imaging inverse problems from a statistical point of view due to their popularity and ability to incorporate nondeterministic information in the forward operator $\boldsymbol{A}$. Indeed, one should think the unknown image $\boldsymbol{x}$ as a realization of the random variable $\mathbf{X}$ to stress the intrinsic uncertainty of this value, which is exaggerated by the potential approximation of the forward operator $\boldsymbol{A}$ that is linked to the measurement process itself. The prior knowledge about $\mathbf{X}$ are captured by the prior probability density function (pdf): $P(\boldsymbol{x})$.
In a similar way, the observed image $\boldsymbol{y}$ can be seen as the realization of a random variable $\mathbf{Y}$, where the likelihood pdf $P(\boldsymbol{y}|\boldsymbol{x})$ describes its behaviour given a fixed $\boldsymbol{x}$.
In this context, the aim is to reconstruct the distribution of $\boldsymbol{x}$ given the observation $\boldsymbol{y}$ and a fixed forward operator $\boldsymbol{A}$.
From a statistical point of view, the objective is to seek the *posterior* pdf $P(\boldsymbol{x} \mid \boldsymbol{y})$. Indeed, using the Bayes formula it is possible to obtain

$$P(\boldsymbol{x}|\boldsymbol{y}) = \frac{P(\boldsymbol{y}|\boldsymbol{x})P(\boldsymbol{x})}{P(\boldsymbol{y})}, \tag{1.3}$$

where $P(\boldsymbol{y})$ is often referred to the evidence term and it should be considered a normalization constant.
Is it possible to switch from this purely statistical to a more optmization-based framework using a maximum a posteriori (MAP) estimation.
The MAP estimation is used to seek an unknown quantity based on observed data. It aims to find the most probable value of a random variable given the observed data and prior knowledge. The formulation became

$$\mathbf{x}^* \in \arg\max_{\mathbf{x}} \; P(\mathbf{x}|\mathbf{y}, \mathbf{A}), \tag{1.4}$$

where applying Bayes formula we obtain:

$$\boldsymbol{x}^* \in \arg\max_{\boldsymbol{x}} \frac{P(\boldsymbol{x})P(\boldsymbol{y}|\boldsymbol{x},\boldsymbol{A})}{P(\boldsymbol{y})},$$

in which after have neglected the evidence term $P(\boldsymbol{y})$ and took the negative likelihood it is possible to get

$$\boldsymbol{x}^* \in \arg\min_{\boldsymbol{x}} -\ln(P(\boldsymbol{x})P(\boldsymbol{y}|\boldsymbol{x},\boldsymbol{A})).$$

Note that some considerations can be done. Assuming that

$$P(\boldsymbol{x}) \propto e^{-R(\boldsymbol{x})}, \; P(\boldsymbol{y}|\boldsymbol{x},\boldsymbol{A}) \propto e^{-\Phi(\boldsymbol{A}\boldsymbol{x};\boldsymbol{y})},$$

the last formulation became

$$\Rightarrow \; \boldsymbol{x}^* \in \arg\min_{\boldsymbol{x}} \; R(\boldsymbol{x}) + \Phi(\boldsymbol{A}\boldsymbol{x};\boldsymbol{y}).$$

We can now define $P(\boldsymbol{x})$ and so $R(\boldsymbol{x})$ to encode some prior information on the ground truth $\boldsymbol{x}$, while $P(\boldsymbol{y}|\boldsymbol{x},\boldsymbol{A})$ and so $\Phi(\boldsymbol{A}\boldsymbol{x};\boldsymbol{y})$ encode some model noise statistics.
$\Phi(\boldsymbol{A}\boldsymbol{x};\boldsymbol{y})$ and $R(\boldsymbol{x})$ are the aforementioned in (1.2) data fidelity and regularisation term.

1.3 Data fidelity

The data fidelity term is a function that quantify the discrepancy between $\boldsymbol{x}$ and $\boldsymbol{y}$. In many real applications the most common discrepancy measure is the classical choice where $\mathcal{D}(\boldsymbol{x},\boldsymbol{y}) = \|\boldsymbol{x}-\boldsymbol{y}\|_2^2$ i.e. least square problem.
We are gonna see that this choice is the preferable one from a Bayesian point of view when our images are corrupted by Gaussian noise.
However, to address types of noise other than Gaussian, different data fidelity terms are required. In this work, we focus on Gaussian and Poisson noise models.

1.3.1 Gaussian data fidelity

Given the assumption that the data are corrupted by Gaussian noise and Gaussian blur, one can think the corrupted observation $\mathbf{y}$ as:

$$\boldsymbol{y} = \boldsymbol{A}\boldsymbol{x} + \boldsymbol{e},$$

where the additive white Gaussian noise is a realization of a Gaussian distribution $\boldsymbol{e} \sim \mathcal{N}(\mathbf{0}, \sigma^2_{AWGN}\boldsymbol{I}_n)$ and $\boldsymbol{A} \in \mathbb{R}^{n\times n}$ is a convolution operator with a Gaussian kernel that models Gaussian blur. The rows of $\boldsymbol{A}$ identify the point spread function (PSF) with a blur variance of σ^2_{PSF}.
In the context of inverse imaging problems, $\boldsymbol{x}$ is the clean image to be estimated, while the measured image $\boldsymbol{y}$ is a realization of the random vector variable $\boldsymbol{Y} \sim \mathcal{N}(\boldsymbol{A}\boldsymbol{x}, \sigma^2\boldsymbol{I}_n)$.
Assuming that the components of the Gaussian noise $\boldsymbol{e}$ are independent. Is it possible to

express the pdf of the components y_i of random vector $\boldsymbol{Y}$ using as parameter the mean $\boldsymbol{Ax}$:

$$f_{\boldsymbol{x},\boldsymbol{A}}(y_i) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(\frac{-(y_i - (\boldsymbol{Ax})_i)^2}{2\sigma^2}\right),$$

and the likelihood of $\mathcal{L}(x|y)$ as

$$\mathcal{L}(\boldsymbol{x}|\boldsymbol{y}) = \prod_{i=1}^{n} f_{\boldsymbol{x}}(y_i) = \prod_{i=1}^{n} \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(\frac{(y_i - (\boldsymbol{Ax})_i)^2}{2\sigma^2}\right) = (2\pi\sigma^2)^{-\frac{n}{2}} \exp\left(-\frac{\|\boldsymbol{y} - \boldsymbol{Ax}\|^2}{2\sigma^2}\right),$$

where n is the number of pixel in the image.
Maximising the likelihood we obtain

$$\arg\max_x \log\left(\mathcal{L}(\boldsymbol{x}|\boldsymbol{y})\right) = \arg\min_x -\log\left(\mathcal{L}(\boldsymbol{x}|\boldsymbol{y})\right) = \arg\min_x \frac{\|\mathbf{y} - \mathbf{Ax}\|_2^2}{2}, \tag{1.5}$$

and the typical data fidelity for Gaussian denoising:

$$\mathcal{D}(\boldsymbol{Ax}, \boldsymbol{y}) = \frac{1}{2}\|\boldsymbol{Ax} - \boldsymbol{y}\|_2^2. \tag{1.6}$$

1.3.2 Poisson data fidelity

Unlike Gaussian noise, Poisson noise is an intensity-dependent noise. There are applications where noise is not independent of the received signal - as assumed so far in the Gaussian case - but depends on the amount of the average quantity of the signal $\mathbf{x}$ and it is defined as follow:

Definition 1.2 (Poisson distribution) *A scalar random variable X is Poisson-distributed with scale parameter λ, denoted by $X \sim Poisson(\lambda)$. The pdf of X takes the form*

$$Poisson_\lambda(X = n) = \frac{\lambda^n}{n!} e^{-\lambda} \quad s.t\ n \in \mathbb{N}.$$

We define a Poisson noisy image components as

$$y_i \sim \text{Poisson}(\mathbf{Ax})_i, \tag{1.7}$$

and for each component $i = 1, \ldots, n$ where $(\mathbf{Ax})_i > 0$, the probability distribution is given by

$$\text{Poisson}_{(\mathbf{Ax})_i}(y_i = n) = \frac{(\mathbf{Ax})_i^n}{n!} e^{-(\mathbf{Ax})_i} \quad \text{s.t } n \in \mathbb{N}. \tag{1.8}$$

Considering the density of the probability distribution as the joint distribution of $\mathbf{Y} \sim \text{Poisson}(\mathbf{Ax})$ as

$$\text{Poisson}_{(\mathbf{x})}(Y = y) = \prod_{i=1}^{n} \frac{(\mathbf{Ax})_i^{y_i}}{y_i!} e^{-(\mathbf{Ax})_i}, \tag{1.9}$$

and assuming the signal $\tilde{y}$ is received, the likelihood is given by

$$\mathcal{L}(x, \tilde{y}) = \prod_{i=1}^{n} \frac{(\mathbf{Ax})_i^{\tilde{y}}}{\tilde{y}_i!} e^{-(\mathbf{Ax}_i)}.$$

The log-likelihood is then

$$\log(\mathcal{L}(\mathbf{x}, \tilde{\mathbf{y}})) = \sum_{i=1}^{n} \tilde{y}_i \log((\mathbf{Ax})_i) - \log(\tilde{y}_i!) - (\mathbf{Ax})_i,$$

and the maximum likelihood approach is equivalent to minimizing the Kullback-Leibler(KL) divergence, obtaining as data fidelity for Poisson noise

$$\mathcal{D}(\boldsymbol{Ax}, \tilde{\boldsymbol{y}}) := \mathrm{KL}(\tilde{\mathbf{y}}, \mathbf{Ax}) = \sum_{i=1}^{n} \tilde{y}_i \log\left(\frac{\tilde{y}_i}{(\mathbf{Ax})_i}\right) + (\mathbf{Ax})_i - \tilde{y}_i. \tag{1.10}$$

The KL divergence is the data fidelity suited for denoising task in the Poisson case.
Notice that in our experiments for the sake of simplicity we do not consider the blurring operator for the Poisson case. In that cases A would be the identity matrix I.

1.4 Regularisation

We recall from definition 1.1 that usually ill-posed problems lack what is the so-called stability, where solving the inverse formulation can cause small perturbations (e.g. additive white Gaussian noise) in the observed image (input) to lead to significant perturbations in the desired image (output).
For example if we want to use a closed-form expression for solving (1.5) is it possible to define the least square term associated to the system $\boldsymbol{y} = \boldsymbol{Ax}$ as

$$J(\boldsymbol{x}) = \frac{1}{2} \|\boldsymbol{Ax} - \boldsymbol{y}\|_2^2,$$

and it possible to find the solution of the minimization problem solving for $\nabla J(\boldsymbol{x}) = 0$, where the solution is defined as

$$\nabla J(\boldsymbol{x}) = \boldsymbol{A}^*(\boldsymbol{Ax} - \boldsymbol{y}),$$

$$\nabla J(\boldsymbol{x}) = 0 \text{ for } \boldsymbol{x} = (\boldsymbol{A}^*\boldsymbol{A})^{-1}\boldsymbol{A}^*\boldsymbol{y}. \tag{1.11}$$

. In Fig 1.1 it has been shown that when only the blur operator is applied to the image, solving the inverse problems lead to good reconstruction. While solving the inverse problem where it has been also applied an negligible quantity of Gaussian noise (e.g. variance of 0.1 in a range of 0-255) will lead to an huge amplification of the noise like in Fig 1.2

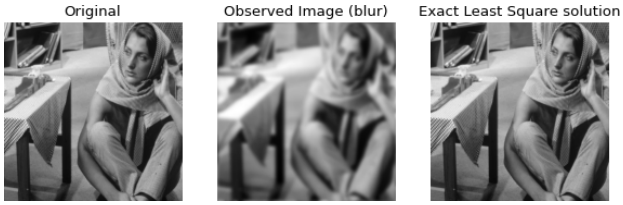


Figure 1.1: Solution of exact square solution with only blur.

To overcome this adversities it is necessary to add in the minimisation problem a regularisation term that ensure some good properties in the solution found.

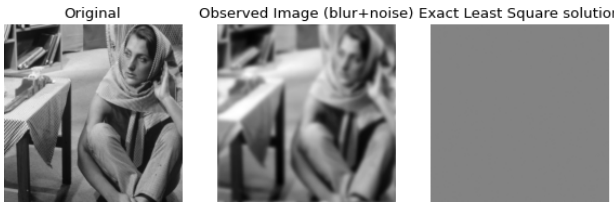


Figure 1.2: Solution of exact square solution with blur and small noise added.

The most common regularisation terms used, especially in regression task for machine learning, are the ℓ_1 norm or the ℓ_2 squared norm.
The ℓ_1 norm it is known for its tendency to produce sparse solution, where many elements of the solution are shrinked to zero, effectively selecting only a subset of the most important features.
The ℓ_1 norm for discrete finite-dimensional elements $\boldsymbol{x} \in \mathbb{R}^n$ is defined as

$$\ell_1(\boldsymbol{x}) := \|\boldsymbol{x}\|_1 = \sum_{i=1}^{n} |x_k|, \tag{1.12}$$

while the l_2 norm squared for discrete finite-dimensional elements $\boldsymbol{x} \in \mathbb{R}^n$ is defined as

$$\ell_2(\boldsymbol{x}) := \|\boldsymbol{x}\|_2^2 = \sum_{i=1}^{n} |x_k|^2. \tag{1.13}$$

Compared to the ℓ_1 norm, the ℓ_2 norm squared (Tikhonov regularisation) shrinks all elements uniformly but does not set any of them exactly to zero, leading to more smooth solutions.
A widely used edge-preserving regulariser initially introduced by Rudin et al. [1992] for image denoising is the Total Variation (TV) semi-norm, which after have added an arbitrary small parameter $0 < \epsilon \ll 1$ for guaranteeing convexity and differentiability. It reads

$$\mathrm{TV}_\epsilon(x) := \sum_{i=1}^{n} \|(\boldsymbol{D}\boldsymbol{x})_i\|_{2,\epsilon} = \sum_{i=1}^{n} \sqrt{(\boldsymbol{D_h}\boldsymbol{x})_i^2 + (\boldsymbol{D_v}\boldsymbol{x})_i^2 + \epsilon^2}, \tag{1.14}$$

where D_h and D_v represent the horizontal and vertical discrete gradient components at pixel i, respectively. The edge-preserving behaviour is given by the presence of the gradient in the regularisation.

To solve the inverse problem given the minimisation formulation given in (1.2) iterative methods have to be applied. In these examples we have used standard gradient descent or proximal gradient descent.
The degredation process illustrated in Fig 1.3 consist in taking the clean image and apply a blurring operator and then injecting a gaussian noise. In Fig 1.4 the minimisation problem has been solved with no regularisation at all. While in Fig 1.5 are illustrated from the left to right different regularisation effect: ℓ_1 norm, ℓ_2 squared norm and TV norm.

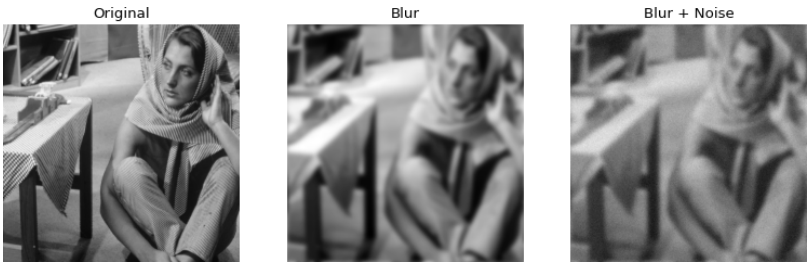


Figure 1.3: Transition from clean image to corrupted.

Figure 1.4: Solution of iterative methods with only L2 Fidelity term.

Figure 1.5: Different effects on the solution based on the regulariser used.

Note that in this case all the regularisation are not weighted by any hyperparameter. Since TV norm is a strong regulariser, not assign the right weight can lead to cartoon-like effect.

1.5 Optimization

Iterative methods are mathematical procedure that given a starting point $\boldsymbol{x}_0$ progressively approximate the sequence $\boldsymbol{x}_k$, ideally converging to an accurate solution. They are commonly used when direct methods lead to ill-posed solutions.
Take for example

$$\min_{\boldsymbol{x}\in\mathbb{R}^n} f(\boldsymbol{x}),$$

with $f : \mathbb{R}^n \to \mathbb{R}$ is a (non)convex, differentiable and proper function. The idea is to find the sequence $\boldsymbol{x}^*$ that minimise f.

1.5.1 Gradient Descent

The most popular algorithm is the gradient descent method where the sequence moves in the opposite direction of the gradient. It moves in the steepest descent with an appropriate

step-size τ.

Algorithm 1 Gradient Descent (GD) algorithm

- **Input:** $\tau \in \left(0, \frac{2}{L}\right), \boldsymbol{x}_0 \in \mathbb{R}^n$.
- **For** $k \geq 0$ **do**
$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \tau \nabla f(\boldsymbol{x}_k)$$
- **End for**

There are several things to consider when iterative methods are used:

- The choice of the step-size τ is important for the convergence of the algorithm and its value is related to the Lipschitz constant L of ∇f, and so on the growth of f.
One example is the case of
$$f(\boldsymbol{x}) = \boldsymbol{x}^2/2 \text{ with } \nabla f(\boldsymbol{x}) = \boldsymbol{x},$$
and the following update rule
$$\boldsymbol{x}_{k+1} = (1 - \tau)\boldsymbol{x}_k.$$
It is clear that an appropriate τ has to be picked for this sequence to converge. For GD algorithm $\tau \in \left(0, \frac{2}{L}\right)$.

- If the function f is convex the solution does not depend on the starting point $\boldsymbol{x}_0$. If this is not the case, different initialisation lead to different solution.
- Ideally the algorithm should stop when the sequence $\boldsymbol{x}_k$ is close to the solution $\boldsymbol{x}^*$ so a stopping criterion has to be defined.
One example is the relative error in the sequence $\|\boldsymbol{x}_{k+1} - \boldsymbol{x}_k\|_1 < \text{tol}$, or gradient check like $\|\nabla f(\boldsymbol{x}_k)\|_1 < \text{tol}$.

1.5.2 Accelerated gradient descent

Nowdays, computational efficiency is necessary, and other first-order methods aim to address this need.
Accelereted gradient descent (AGD) algorithm proposed by Nesterov [1983] provides faster convergence and the idea is to add inertia to "shift" the sequence of iterates (Fig 1.6).

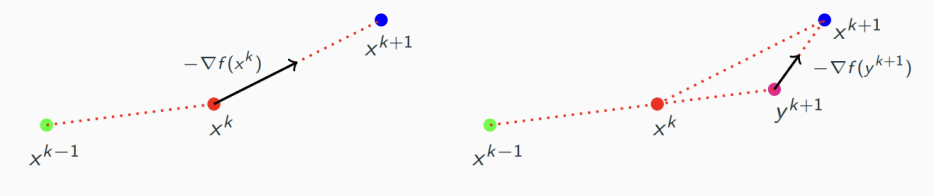


Figure 1.6: Difference between GD and AGD directions.

Algorithm 2 Accelerated gradient descent (AGD) algorithm

- **Input:** $\boldsymbol{x}_0 = \boldsymbol{x}_{-1} \in \mathbb{R}^n$, $\tau \in (0, \frac{1}{L}]$, $t_0 = 1$.
- **For** $k \geq 0$ **do**

$$t_{k+1} = \frac{1 + \sqrt{1 + 4t_k^2}}{2}$$
$$\boldsymbol{y}_{k+1} = \boldsymbol{x}_k + \frac{t_k - 1}{t_{k+1}}(\boldsymbol{x}_k - \boldsymbol{x}_{k-1})$$
$$\boldsymbol{x}_{k+1} = \boldsymbol{y}_{k+1} - \tau \nabla f(\boldsymbol{y}_{k+1})$$

- **End for**

1.5.3 Fast iterative shrinkage-thresholding algorithm (FISTA)

Some problems require the solution $\boldsymbol{x}^*$ to be included in a specific suitable domain, assigning to the solution some costraints. For example in Poisson noise denoising the solution must be included in $\mathbb{R}^n_{++}$.
For these reason sometimes is necessary to use proximity method like Fast iterative shrinkage-thresholding algorithm(FISTA) based on the work of Beck and Teboulle [2009] and it is defined as follow

Algorithm 3 Fast Iterative Shrinkage-Thresholding Algorithm (FISTA)

- **Input:** $\boldsymbol{x}_0 = \boldsymbol{x}_{-1} \in \mathbb{R}^n$, $\tau \in (0, \frac{1}{L}]$, $t_0 = 1$, with $\mathbb{D}$ a feasible set.
- **For** $k \geq 0$ **do**

$$
\begin{aligned}
t_{k+1} &= \frac{1 + \sqrt{1 + 4t_k^2}}{2} \\
\boldsymbol{y}_{k+1} &= \boldsymbol{x}_k + \frac{t_k - 1}{t_{k+1}}(\boldsymbol{x}_k - \boldsymbol{x}_{k-1}) \\
\boldsymbol{x}_{k+1} &= \text{prox}_{\mathbb{D}} \boldsymbol{y}_{k+1} - \tau \nabla f(\boldsymbol{y}_{k+1})
\end{aligned}
$$

- **End for**

1.6 Image processing metrics

The most used metrics in image processing are the structural similarity index measure (SSIM) and the peak signal-to-noise ratio (PSNR).
PSNR is a metric that quantifies the difference between two images by comparing the peak signal (original image) to the noise (distortions or differences). It is expressed in decibels (dB) and it is formulated as

$$
\text{PSNR} = 20 \log_{10} \left(\frac{\max(\boldsymbol{x})}{\sqrt{MSE(\boldsymbol{x}, \tilde{\boldsymbol{x}})}} \right),
$$

where $\boldsymbol{x}$ is the ground truth (also the clean image) and MSE is the minimum squared error pixel-wise between the ground truth and the reconstructed image $\tilde{x}$.
SSIM is a perceptual metric that measures the similarity between two images, focusing on structural information rather than just pixel differences (unlike PSNR). It produce a score in the range [0,1].
The SSIM metric is defined as follow:

$$
\text{SSIM}(\boldsymbol{x}, \boldsymbol{y}) = \frac{(2\mu_x \mu_y + c_1)(2\sigma_{xy} + c_2)}{(\mu_x^2 + \mu_y^2 + c_1)(\sigma_x^2 + \sigma_y^2 + c_2)}
$$

with:

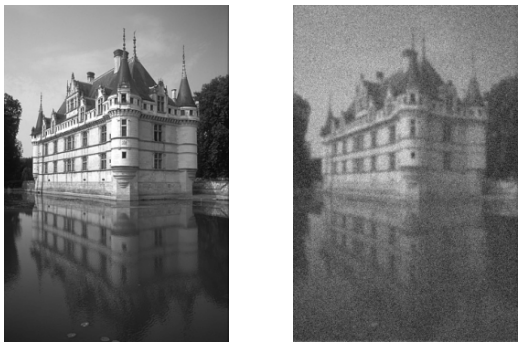
- μ_x: the pixel sample mean of x, μ_y: the pixel sample mean of y.
- σ_x^2: the variance of x, σ_y^2: the variance of y.
- σ_{xy}: the covariance of x and y;
- $c_1 = (k_1 L)^2$, $c_2 = (k_2 L)^2$.
- L: the dynamic range of the pixel-values, often referred as $\max(\boldsymbol{x})$
- $k_1 = 0.01$ and $k_2 = 0.03$ by default.

Regarding both the metrics, the higher they are and closer the reconstructed image is to the clean image.

Chapter 2

Hyperparameters estimation

Recalling (1.2), a suitable scalar hyperparameter (or vector of hyperparameters) have to be found to balance the influence of the data fidelity and regularisation terms, that is the balance between how much we trust the data and our prior knowledge on the structure of the desired solution.
Firstly, it will be discussed the scalar case, which will be referred to as space-invariant (or global) approach, where each region of the image is regularised in the same way due the estimation of a single regularisation parameter weighting the global strength of the data term against the global effect of regularisation. Then we will describe a space-variant approach where a parameters map is computed so that each pixel is assigned with its respective hyperparameter balancing the *local* effect of regularisation and data fit. Recalling the use of TV regularisation term favouring piecewise constant regions, the rationale behind a space-variant choice is to regularise less the textured regions of the image and to regularise more the homogeneous region, see for example the castle details and the sky in 2.1
Indeed in Fig 2.2 we show the differences in the reconstruction for under-and over-estimated



(a) $\boldsymbol{x}$ (b) $\boldsymbol{y}$

Figure 2.1: Ground truth image and its noisy version. Noise is Gaussian with $\sigma^2_{AWGN} = 0.1$ and Gaussian blur with $\sigma^2_{PSF} = 1.5$.

global hyperparameter for TV reconstruction — a too small value will over-smooth the image, while a too big value will not remove the noise at all. The task thus consist in choosing the optimal one which will try to find a suitable balance.

(a) $\tilde{\boldsymbol{x}}$, $\mu = 5$ (b) $\tilde{\boldsymbol{x}}$, $\mu = 50$

Figure 2.2: Different reconstruction obtained given by the choice of μ.

2.1 Space invariant regularisation in the Gaussian case

The inverse problem that we are treating in the Gaussian case has the form of

$$\boldsymbol{y} = \boldsymbol{A}\boldsymbol{x} + \boldsymbol{e}$$

where $\boldsymbol{A} \in \mathbb{R}^{n\times n}$ models Gaussian blur defined by a kernel variance of σ^2_{PSF} and $\mathbf{e} \sim \mathcal{N}(\mathbf{0}, \sigma^2_{AWGN}\boldsymbol{I}_n)$ is a random realization of additive white Gaussian noise (AWGN).
The typical framework for image reconstruction is

$$\tilde{\boldsymbol{x}}_\mu = \arg\min_x \mu\mathcal{D}(\boldsymbol{A}\boldsymbol{x}, \boldsymbol{y}) + \mathcal{R}(\boldsymbol{x}). \tag{2.1}$$

Recalling the data fidelity for Gaussian denoising in (1.6) and the TV norm regulariser in (1.14) we define the variational framework for Gaussian denoise as

$$\tilde{\boldsymbol{x}}_\mu = \arg\min_x \frac{\mu}{2} \|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{y}\|_2^2 + \mathrm{TV}_\epsilon(\boldsymbol{x}). \tag{2.2}$$

We remark that, usually the hyperparameter is situated in front of the regulariser(hence it is referred to as *regularisation parameter*), but we preferred to put it in front of the data fidelity for numerical reasons (i.e. untreatable small value).[1]

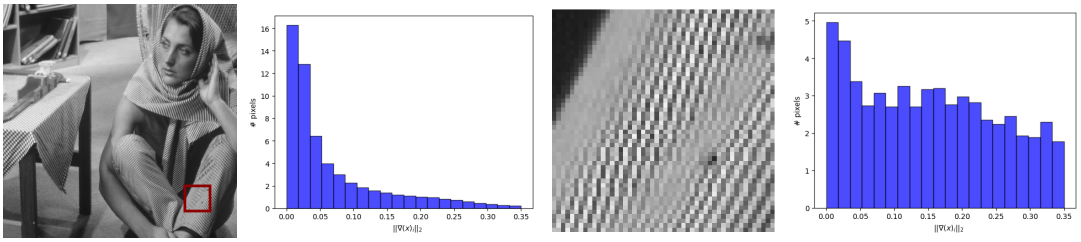
Definition 2.1 (Univariate exponential or half Laplacian distribution) *A scalar random variable X is exponential- or half Laplacian-distributed with scale parameter $\lambda \in \mathbb{R}_+/\{0\}$, denoted by $X \sim hL(\lambda)$, if $X = |Y|$, with $Y \sim L(0, \lambda)$. The pdf of X takes the form*

$$\pi_\lambda(x) = \begin{cases} \lambda \exp(-\lambda x) & \text{if } x \in \mathbb{R}_+, \\ 0 & \text{otherwise.} \end{cases}$$

As mentioned in the work of Pragliola et al. [2023], a TV prior assumes that the gradient

[1] The scale of parameters in front of the TV norm typically falls within the range of $[10^{-3}, 10^{-2}]$, which can pose numerical challenges in machine learning algorithms, especially those involving regression task.

magnitudes $q_i := \|(\mathbf{Dx})_i\|_2$ follow a space-invariant one-parameter half-Laplacian Distribution (hLD) with global scale parameter, which does not adapt to local image structures.
Indeed the prior hLD probability density in natural images is highly dependent on the region of the image that it is taken in consideration. Indeed, the parameter λ defining the shape of



(a) $\mathbf{x}$ (b) Hist of $\|(\mathbf{Dx})_i\|_2$ (c) Detail of $\mathbf{x}$ (d) Hist of $\|(\mathbf{Dx})_i\|_2$

Figure 2.3: Global and local image content prior probability.

the distribution is dependent on the region, as shown in Fig 2.3
Whenever the ground truth version of the noisy image $\boldsymbol{y}$ is available the optimal parameter μ can be determined by maximizing an appropriate metric that measures similarity between the ground truth(GT) and the reconstructed image.

To address the task of optimal hyperparameter estimation, a natural formulation is the following grid search optimization problem

$$\begin{cases} \arg\max_{\mu} \ell(\tilde{\boldsymbol{x}}_{\mu}, \boldsymbol{x}_{\mathrm{GT}}), \\ \text{subject to } \tilde{\boldsymbol{x}}_{\mu} = \arg\min_{\boldsymbol{x}} \ \mu\mathcal{D}(\boldsymbol{A}\boldsymbol{x}, \boldsymbol{y}) + \mathcal{R}(\boldsymbol{x}). \end{cases} \tag{2.3}$$

For our experiments on Gaussian denoising, the regulariser $\mathcal{R}$ is the TV norm presented in (1.14) and the data fidelity $\mathcal{D}$ is the one presented in (1.6).
The $\ell(\cdot)$ is a loss function assessing optimality that can be chosen, for instance, either as the PSNR or the SSIM.
The metric used for our experiments is the $\ell(\tilde{\boldsymbol{x}}_{\mu}, \boldsymbol{x}_{\mathrm{GT}}) = \mathrm{SSIM}(\tilde{\boldsymbol{x}}_{\mu}, \boldsymbol{x}_{\mathrm{GT}})$.
Solving directly (2.3) is challenging as shown in Calatroni et al. [2017], Calatroni et al. [2020] and Calatroni et al. [2019]. Hence, practically, for this scalar case we decided to use instead a brute-force approach, searching for the optimal parameter within a specific range of tentative hyperparameters.
In Fig 2.4 we plot the function $f(\mu) = \mathrm{SSIM}(\boldsymbol{x}_{GT}, \boldsymbol{x}_{\mu})$ for several values of μ within a fixed interval. We observe that this function is concave, with a clear optimal value $\hat{\mu}$ that maximise the SSIM metric.

To avoid the need of exploring, for different images and noise levels, the whole set of grid points , we implement a *golden-section* algorithm, also called golden-section search, reducing drastically the computation cost required to compute the optimal μ. The algorithm starts with a large initial interval $[a, b]$ and then, step by step, reduces its size until it becomes sufficiently small.
The following steps are required:

1.
   - Choosing an interval $[a, b]$ ideally containing the optimal μ
   - Assign an initial centroid μ^*, typically $\mu^* = \frac{a+b}{2}$
   - Define a vector of decreasing ranges $\alpha = [\alpha_1, \alpha_2, ..., \alpha_n]$

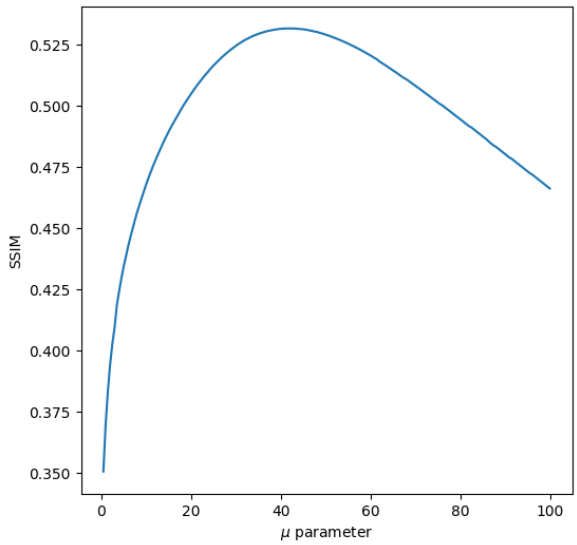


Figure 2.4: Brute force approach for computing the optimal parameter.

2. Evaluate the function in the triplet $[\mu^* - \alpha_i, \mu^*, \mu^* + \alpha_i]$
3. Based on the function values, select as a new centroid the value that obtained the best result.
4. Repeat until the granularity given by the α_i is sufficiently small.

We show the granularity of the parameter search in the dot plot in Fig 2.5

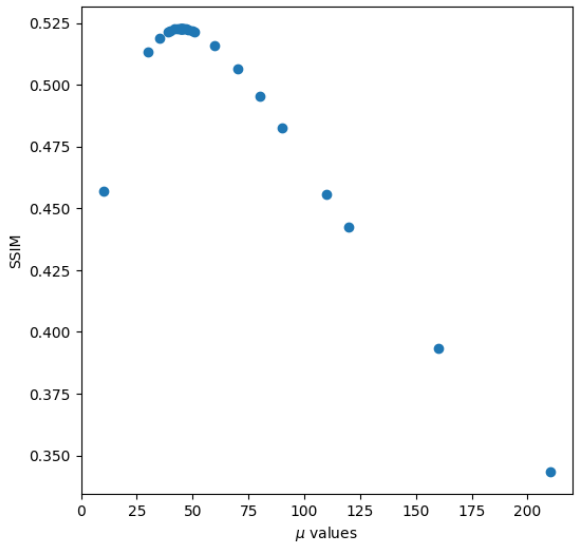


Figure 2.5: Dot plot of the value examined by golden-section algorithm.

2.1.1 Algorithmic settings

In this section, we describe the algorithms employed to solve the ℓ_2-TV problem in an efficient way.

$$f(x) = \frac{\mu}{2} \|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{y}\|_2^2 + TV_\epsilon(\boldsymbol{x}),$$

$$\nabla f(x) = \mu \boldsymbol{A}^*(\boldsymbol{A}\boldsymbol{x} - \boldsymbol{y}) - \operatorname{div}\left(\frac{\boldsymbol{D}\boldsymbol{x}}{\|\boldsymbol{D}\boldsymbol{x}\|_{2,\epsilon}}\right),$$

Note that the standard TV regularisation term is non-smooth and as such it does not allow an explicit computation of the gradient, thus requiring more sophisticated algorithms for its solution as shown in Chambolle and Pock [2011]. However, thanks to the ϵ-smoothing performed here, a gradient exists and efficient gradient-based solvers can be used. The algorithm used is FISTA (Section 1.5.3), ensuring that the minimization problem is solved in a accelerated way while keeping the solution in the range $[0,1]^n$. To guarantee that the final sequence $\boldsymbol{x}_k$ remains comparable to the same range as the initial image $\boldsymbol{x}$.
The following parameter choices for the algorithm have been made for the following tests.

- The stopping condition on the relative error with respect to the cost function $\frac{|f(\boldsymbol{x}_k)-f(\boldsymbol{x}_{k-1}|)}{f(\boldsymbol{x}_k)} < 10^{-5}$ or when the maximum of the iteration (usually 500) is reached.
- The ϵ present in the TV regulariser for guaranteeing differentiability is setted as 10^{-2}
- The action of the linear operator, $\boldsymbol{A}$ or $\boldsymbol{A}^*$ is defined as the convolution of a gaussian kernel and it has been applied using the 2D Fast Fourier Transform (FFT)
- The gradient of the smoothed TV term is computed using by G.Peyre library.
- After each iteration the sequence $\boldsymbol{x}_k$ is projected into the convex set $[0,1]^n$.

We need to set $\tau = \frac{1}{L}$ with L defined as the Lipschitz constant of ∇f.
By triangle inequality, this L can be estimated as the sum of the Lipschitz constants of the data and regularisation terms for data fidelity and regularisation.

$$L = \mu \left\|A^T A\right\| + \frac{8}{\epsilon}.$$

In our case, A is a convolution operator and the term $\left\|A^T A\right\|$ is equal to the maximum of the Fourier transform associated to the kernel filter, so $\left\|A^T A\right\| = 1$.
Note that the stopping condition is based on the relative error of the cost function rather than the relative error of the sequence norm, because FISTA may show oscillations in the sequence of iterates.

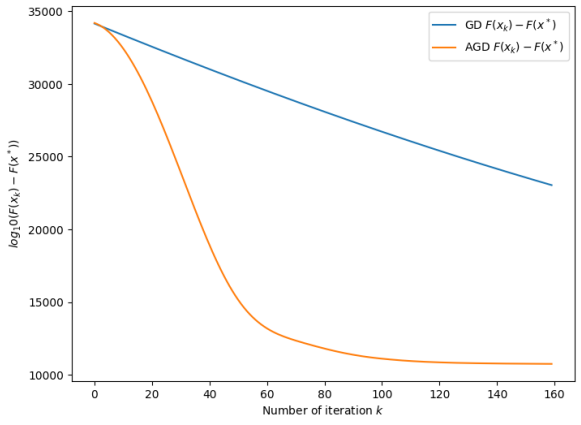
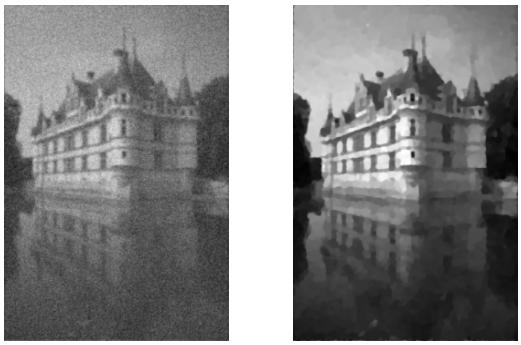


Figure 2.6: Different rate of convergence between GD and FISTA.



(a) GD (b) FISTA

Figure 2.7: Solution obtained by the solvers after 160 iterations.

To assess the correct implementation of the algorithm and to compare it with standard (non-accelerated) gradient descent, using as example the deblurring task with $\sigma^2_{AWGN} = 0.1, \sigma^2_{PSF} = 1.0$, we compared both the performances of GD and FISTA in term of computational time to compute $\boldsymbol{x}^*$ and the iterations required.
The GD converged in 66.56 seconds with 1247 iterations, while FISTA converged in 12.73 seconds with 196 iterations.
Fig 2.6 and 2.7 illustrates that, with an upper limit of 160 iterations, the solution obtained by FISTA has nearly converged, whereas the standard GD solution remains far from convergence. Furthermore, important considerations need to be done. The time required to compute the optimal hyperparameter presented in (2.3) using the golden-section algorithm vary based on the presence of Gaussian blur or the intensity of Gaussian noise. Indeed, images with higher noise intensity or images affected by blur require more iterations of the underlying optimization solvers to arrive at convergence.
Taking as example an image of size 321×481 we obtain the following computational time required for optimal parameter image reconstruction

- $A = I$, $\sigma^2_{AWGN} = 0.01 : \sim 92$s
- $A = I$, $\sigma^2_{AWGN} = 0.02 : \sim 111$s
- $\sigma^2_{PSF} = 1.0$, $\sigma^2_{AWGN} = 0.01 : \sim 195$s
- $\sigma^2_{PSF} = 2.0$, $\sigma^2_{AWGN} = 0.01 : \sim 169$s

All the codes are publicy avaiable at the following GitHub repository[2].

2.2 Scalar regularisation in the Poisson case

We now consider the case when signal-dependent Poisson noise corrupts the observed image, where each component is defined as:

$$y_i \sim \text{Poisson}(x_i + \eta) \tag{2.4}$$

[2] https://github.com/ClaudioFantasia/Space-Variant-Hyperparameter-Estimation-for-Inverse-Imaging-Problem

with η a small background term to ensure that the Poisson distribution is always defined even when $x_i = 0$.

Recalling the Def 1.2, Poisson noise takes the pixel's intensity as parameter of a Poisson distribution (i.e. $\lambda = x_i$), and outputs a new value based on the probability associated with that intensity.
That is, for each $i = 1, ..., n$ where $(x_i \geq 0)$. Each component $y_i \sim \text{Poisson}(\alpha x_i + \eta)$ has a probability distribution given by

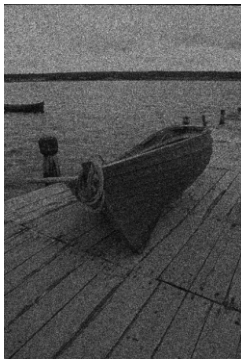
$$\pi_{\alpha x_i+\eta}(y_i = n) = \frac{(\alpha x_i + \eta)^n}{n!} e^{-\alpha x_i+\eta} \quad \text{s.t } n \in \mathbb{N}, \tag{2.5}$$

where α is a parameter that characterise how intense the noise is. In our case $(0 \leq x \leq 1)$, the smaller is alpha, the more intense is the noise.
Note that a random variable $X \sim \text{Poisson}_{\alpha x_i+\eta}(n)$ has $\mathbb{E}(X) = Var(X) = \alpha x_i + \eta$. So white pixels (i.e. higher value) have a higher probability of "oscillation", leading to increased noise in those pixels.
Indeed, an image with a lower mean and overall darker pixels will experience a smaller ℓ_2 error after Poisson noise is applied with a fixed parameter. While a brighter image will experience higher ℓ_2 error, so higher discrepancy between the ground truth and its noisy version. A clear example is shown in Fig 2.8

(a) Ground truth with $\mathbb{E}[\mathbf{x}] = 0.26$

(b) Noisy image $\mathbf{y}$ with $\ell_2(y, x_{GT}) = 46.99$

(c) Ground truth with $\mathbb{E}[\mathbf{x}] = 0.59$



(d) Noisy image $\mathbf{y}$ with $\ell_2(y, x_{GT}) = 122.26$

Figure 2.8: Different image corruption based on average pixel's intensity.

Sticking again with TV regularisation and the data fidelity for Poisson noise presented in Eq (1.14) and Eq (1.10), the variational framework to consider for Poisson denoising is

$$\tilde{\mathbf{x}} \in \arg\min_{\mathbf{x}} \mu \text{KL}_\eta(x, y) + \text{TV}_\epsilon(x), \tag{2.6}$$

where $\mu > 0$ is the hyperparameter to estimate and KL_η is the smoothed Kullback-Leibler divergence

$$\mathrm{KL}_\eta(x, y) = \sum_{i=1}^{n} y_i log\left(\frac{y_i}{x_i + \eta}\right) + x_i - y_i.$$

The smoothed KL fidelity term requires each component to be positive ($x_i \geq 0$). This turns the inverse problem into one of minimizing the KL divergence subject to a positivity constraint $\mathbf{x} \geq 0$.
The variational framework for Poisson denoising is:

$$\tilde{\mathbf{x}} \in \arg\min_{\mathbf{x}} \ \mu \sum_{i=1}^{n} y_i log\left(\frac{y_i}{x_i + \eta}\right) + x_i - y_i + \sqrt{(D^h x_i)^2 + (D^v x_i)^2 + \epsilon^2} \quad \text{s.t } \mathbf{x} > 0. \tag{2.7}$$

For numerical reasons, a parameter η was also included in the KL divergence term to avoid divisions by zero and to be Lipschitz continuously differentiable on $\mathbb{R}^n_+$.

2.2.1 Algorthmic settings

Taking as function

$$f(x) = \sum_{i=1}^{n} y_i log\left(\frac{y_i}{x_i + \eta}\right) + x_i - y_i + \sqrt{(D^h x_i)^2 + (D^v x_i)^2 + \epsilon^2} \quad \text{s.t } \mathbf{x} \geq 0,$$

$$\nabla f(x) = \mu\left(\frac{y}{x + \eta} - 1\right) - \mathrm{div}\left(\frac{Dx}{\|Dx\|_{2,\epsilon}}\right).$$

The function f with parameter $\eta > 0$ and $\epsilon > 0$ is Lipschitz continuously differentiable on $\mathbb{R}^n_+$ with Lipschitz constant

$$L = \mu \frac{\max(y)}{\eta^2} + \frac{8}{\epsilon}.$$

A rigorous proof of the Lipschitz constant for KL term is present in Harmany et al. [2011]. To enforce the non-negativity of the solution, we consider a projection on the non-negative orthant which makes the underlying objective non-smooth. To solve it FISTA was used again as a solver, performing a projection into a convex set [0,1]^n.and the solver settings remains the same with respect to the Gaussian case.
Note that given the estimation of L above, the following dependencies hold for L and the step-size $\tau = 1/L$:

$$L = \mathcal{O}(1/\epsilon^2), \quad \tau = \mathcal{O}(\epsilon^2) \tag{2.8}$$

From a practictal point of view, it is preferable to keep η as small as possible to minimize the discrepancy between the smoothed KL divergence and the true KL divergence, under this choice and looking at (2.8) we have that the step size also tends to zero, making each step of the algorithm too little.
To overcome the problems caused from the lack of a well-defined Lipschitz constant, or the unstable behaviour due to numerical stability, we considered adaptive backtracking techniques favouring a (non-monotone) adjustement of the step-size value.
Traditional Armijo-type line search methods for computing suitable step-sizes were not considered, as they do not allow for local increasing and decreasing of it, which could be potentially unefficient in case of small initialisations. These strategies permit gradual step-size increases along iterations while still ensuring the convergence of the algorithm.

In order to write the optimization step, a standard descent step in the differentiable component f is combined with the projection onto the convex feasible set $[0,1]^n$.
Based on the work of Calatroni and Chambolle [2019] we define the projection onto the convex feasible set as $\hat{x} = \text{prox}_{[0,1]}(x)$ where x is any point included in $\mathbb{R}^n$ and a suitable step-size τ picked based on the following condition

$$D_f(\hat{x}, x) \leq \frac{\|\hat{x} - x\|^2}{2\tau}, \tag{2.9}$$

where the Bregman distance D_f is defined as

$$D_f(x, y) := f(x) - f(y) - < \nabla f(x), x - y > \tag{2.10}$$

The projection is the clipping one and it is achieved by setting all values x_i below zero to zero and all values above zero to zero. The algorithm proposed is the following:

Algorithm 4 FISTA with backtracking

Parameters: $\tau_1 > 0, \rho \in (0, 1)$.
Initialization: $x^0 = x^{-1} \in \mathbb{R}^n$, $x_1 = \text{prox}_{[0,1]}(x_0)$, and $t_1 = 1$.
For $k = 1, 2, \ldots$

$$\tau_{k+1}^0 = \frac{\tau_k}{\rho}; \tag{2.11}$$

For $i = 0, 1, \ldots$ **repeat**

$$\tau_{k+1} = \rho^i \tau_{k+1}^0, \tag{2.12}$$

$$t_{k+1} = \frac{\sqrt{(1 + 4\frac{\tau_k}{\tau_{k+1}} t_k^2)}}{2}, \tag{2.13}$$

$$\beta_{k+1} = \frac{t_k - 1}{t_k + 1}, \tag{2.14}$$

$$y^{k+1} = x^k + \beta_{k+1}(x^k - x^{k-1}), \tag{2.15}$$

$$x^{k+1} = \text{prox}_{[0,1]}\left(y^{k+1} - \tau_{k+1} \nabla f(y^{k+1})\right), \tag{2.16}$$

Until

$$D_f(x^{k+1}, y^{k+1}) \leq \|x^{k+1} - y^{k+1}\|^2 / 2\tau_{k+1} \tag{2.17}$$

As can be seen from Fig (2.9), the step sizes along the iterations not only had a non-monotonic trend, but were always above the step size defined by the Lipschitz constant (red dotted line)

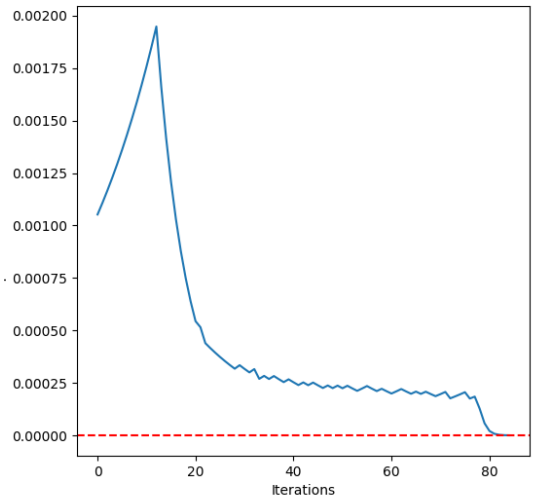


Figure 2.9: Step sizes of FISTA along the iterations.

Although backtracking strategies were implemented, the low Lipschitz constant lead the computational time required for Poisson denoising to be significantly higher compared to Gaussian denoising computational time shown in Section 2.1.1. However, in this case the time required for the solver to converge is not related to the noise intensity.

- $\alpha = 45 : \sim 252$s
- $\alpha = 30 : \sim 252$s
- $\alpha = 15 : \sim 258$s

2.3 Space-variant regularisation

We now describe a strategy to model local image information examining small image patches as showed in the following experiments. This will allow us to "piece together" local information in a way that defines a more effective and appropriate image regularizer

Recalling the Bayesian interpretation of the TV approach in Fig 2.3, we can now allow allow the gradient scale to **locally adapt** to the pixel-dependent structure, which means considering the adaptive prior:

$$\pi_{\lambda_i}(x_i) = \begin{cases} \lambda_i \exp(-\lambda_i x) & \text{if } x_i \in \mathbb{R}_+, \\ 0 & \text{otherwise.} \end{cases} \tag{2.18}$$

A new formulation where a distinct regularisation parameter is estimated for each pixel has been devised. We allow the model to better adapt to local image content and effectively capture the space-variant (SV) nature of the gradient distribution across the image. The

formulation that follows is

$$\tilde{x} \in \arg\min_{x} \sum_{i=1}^{n} \mathcal{D}((Ax)_i, y_i) + \lambda_i \mathcal{R}(x_i), \quad \boldsymbol{\lambda} = (\lambda_i) \in \mathbb{R}^n. \tag{2.19}$$

However due to numerical problems (i.e. untreatable small values for the machine learning task) caused by the strong effect of the TV norm, we preferred to use

$$\tilde{x} \in \arg\min_{x} \sum_{i=1}^{n} \mu_i \mathcal{D}((Ax)_i, y_i) + \mathcal{R}(x_i), \quad \boldsymbol{\mu} = (\mu_i) \in \mathbb{R}^n. \tag{2.20}$$

In the case where the parameter is scalar, it is possible to prove that μ is inversely proportional to λ, using the assumption that both parameters have to strictly positive. However, this is no longer necessarily true when the parameter becomes pixel-wise.
Nevertheless, considering that these parameters are not calculated on a single pixel but - as explained in the following chapter - each pixel-wise parameter is actually first calculated over a sub-image (where the pixel is the centroid), and then the parameter computed for the sub-image is assigned to the pixel.
Considering that we can see each sub-image as a limit case of the scalar case, our substitution remain valid.

For this new space-variant approach, we first introduce a Gaussian pixel-wise fidelity resulting in an adaptive version of (2.2)

$$\tilde{\mathbf{x}} \in \arg\min_{\mathbf{x}} \sum_{i=1}^{n} \frac{\mu_i}{2} |(\mathbf{Ax})_i - y_i|^2 + \mathrm{TV}_\epsilon(\mathbf{x}), \quad \boldsymbol{\mu} = (\mu_i) \in \mathbb{R}^n, \tag{2.21}$$

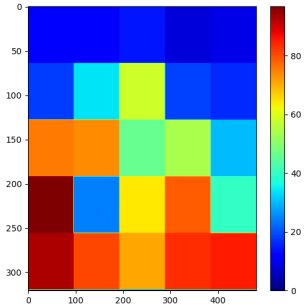
and then a Poisson pixel-wise fidelity, an adaptive version of (2.7)

$$\tilde{\mathbf{x}} \in \arg\min_{\mathbf{x}} \sum_{i=1}^{n} \mu_i \left(y_i log\left(\frac{y_i}{x_i + \eta}\right) + x_i - y_i\right) + \sqrt{(D^h x_i)^2 + (D^v x_i)^2 + \epsilon^2} \quad \text{s.t } \mathbf{x} > 0. \tag{2.22}$$

Using as example the problem of deblurring a Gaussian noisy image, we want to emphasise the effectiveness of the space-variant approach: observing the color bar of the parameters map, it is possible to notice in Fig 2.11b lower values (i.e. more regularisation) in homogeneous areas like the sky, where details are minimal. While in textured regions such as the buffaloes or grass, regularization is reduced and we have higher values.
It is important to note that regularization becomes stronger as the parameter decreases (and viceversa). This is due to the fact that the parameter is in front of the data fidelity.

Figure 2.10: Blurry and noisy image to reconstruct.

(a) Reconstructed image



(b) Color map of the parameters intensity

Figure 2.11: Reconstructed image with its patch-wise parameters map.

Chapter 3

Patch-based learning

We present in the following a patch-based strategy to learn TV parameter maps. To do so, in this chapter we start describing the dataset used in our experiments and summarize some of its key properties. We explain how we simulate noisy images by corrupting the ground truth images as a training set, in order to have both the noisy images and their corresponding clean counterparts. Furthermore, we discuss the motivations that led us to adopt a supervised deep learning approach and how we generated a secondary dataset, where sub-images(i.e. patches) were extracted from the images present in the dataset and their corresponding SSIM-optimal parameter was computed using golden-section methods mentioned in the previous chapter. Patches were created using an overlapping sliding window approach and each parameter is then used as label for the corresponding patch.

3.0.1 Notation

We introduce some notation for reader clarity.

Parameter	**Description**
$(\boldsymbol{x}_{GT}^k)_i$	i-th pixel of the k-th image
$\bar{\boldsymbol{x}}_n^k$	n-th patch of the k-th image
$(\boldsymbol{y}_{GT}^k)_i$	i-th pixel of the k-th corrupted image
$\bar{\boldsymbol{y}}_n^k$	n-th patch of the k-th corrupted image
$(\mu_{DL}^k)_i$	Predicted parameter for the i-th pixel of the k-th image
$(\bar{\mathbf{y}})_i$	Patch centered around i-th pixel
$\mu_{GT}^{k,n}$	Optimal scalar hyperparameter corresponding to the n-th patch of the k-th image
$(\mu_{DL})_i$	Predicted sub-optimal parameter for i-th pixel
$\boldsymbol{\mu}^{GT_d}$	Optimal scalar parameters (i.e. labels) associated to the d-th simulated dataset
f_θ	Neural network with weights θ

Table 3.1: Notation

3.1 Berkley dataset

The dataset used for our simulations is the Berkeley dataset, consisting of 300 greyscale natural images with sizes of either 321×481. These images are real-world scenes of landscapes, people, and objects, with patterns and structures found in nature rather than artificially created. In Fig 3.1 we show some examples and it is possible to found the dataset at the following link[1].

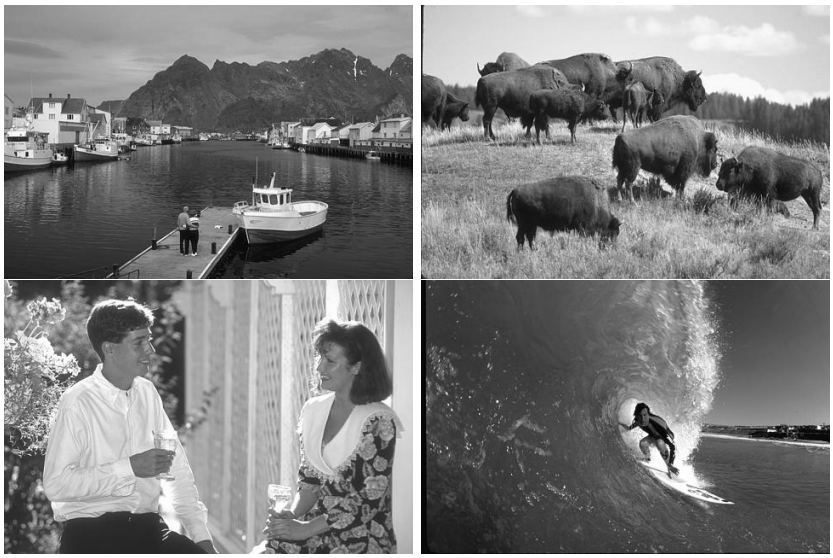


Figure 3.1: Some examples from Berkley dataset.

This kind of images differ significantly in contents, showing several different regions with very different statistics. There could be highly textured region like a person face full of details and more piecewise constant regions like the sky with only very few intensity variations. Having images with so different behaviour can be challenging for the task of hyperparameter estimation, but it may be beneficial from a deep learning perspective enhancing more variety in the dataset and learning of new patterns.
The data range of all the images has been normalized in the interval [0,1].
To simulate realistic corrupted images, we introduced Gaussian noise (with or without additional blur) and Poisson noise.
We added Gaussian noise by summing to the normalized k-th image of the dataset $\boldsymbol{x}_{GT}^k$ another image of the same size $\boldsymbol{e}$, where all pixels follow a Gaussian distribution with a mean of 0 and a variance that could either be fixed or variable.

$$\boldsymbol{y}_{GT}^k = \boldsymbol{x}_{GT}^k + \boldsymbol{e} \quad \boldsymbol{e} \sim \mathcal{N}(0, \sigma_{AWGN}^2 \mathrm{Id}),$$

For simulating blurred and noisy images, we first applied the convolution operator A to the k-th image and Gaussian noise was then added:

$$\boldsymbol{y}_{GT}^k = \boldsymbol{A}\boldsymbol{x}_{GT}^k + \boldsymbol{e} \quad \boldsymbol{e} \sim \mathcal{N}(0, \sigma_{AWGN}^2 \mathrm{Id}).$$

To model Poisson noise, a Poisson distribution was applied to each pixel of the ground truth

[1] https://www2.eecs.berkeley.edu/Research/Projects/CS/vision/bsds/

image, obtaining:

$$(y^k_{GT})_i \sim Poisson((x^k_{GT})_i) \quad \text{s.t. } i \in \{1, \dots, I\},$$

where I is the total number of pixels in the image.

To simulate different noise levels, we considered the following settings: for Gaussian noise, the variance was set to $\sigma^2_{AWGN} = 0.01$ and $\sigma^2_{AWGN} = 0.02$. For blurred images, the blur variance of the point spread function (PSF) was typically $\sigma^2_{PSF} = 1.0$.
For Poisson noise, the noise levels were determined by scaling factors $\alpha = 45, \alpha = 15, \alpha = 5$. Subsequently, the images have been normalized in a fixed data range [0,1], so that the corrupted images $\boldsymbol{y}_{GT}$ and the clean one $\boldsymbol{x}_{GT}$ would be in the same range.
In the Gaussian case we normalised with a *min-max normalisation*

$$\boldsymbol{y}' = \frac{\boldsymbol{y} - \min(\boldsymbol{y})}{\max(\boldsymbol{y}) - \min(\boldsymbol{y})} \tag{3.1}$$

Whereas in the Poisson case we normalised by simply dividing by the maximum, given the strict positivity of each y_i component.

$$\boldsymbol{y}' = \frac{\boldsymbol{y}}{\max(\boldsymbol{y})} \tag{3.2}$$

3.2 Training dataset

We created a dataset based on the Berkeley dataset, with the aim of using it as a training set for a deep learning model.
Out of the $K = 300$ images in the Berkeley dataset. We obtain $N = 126$ (or $N = 551$) patches per image depending on the patches size. Overall the samples in the dataset are $K \times N$, thus they are 37800 or 165300.
In order to extract only relevant features from an image and create an effective dataset, for each corrupted image $\boldsymbol{y}^k_{GT}$ slightly overlapping patches $\bar{\boldsymbol{y}}^k_n$ of size 64×64 (or 32×32) were extracted with a sliding step set to 32 (or 16), either to the right or downward, thus letting the patches to overlap by 50 or 25 percent.
The labels are defined as the scalar hyperparameter appearing in (2.1) of the corresponding sample patch $\bar{\boldsymbol{y}}$ estimated by solving the related grid search (2.3). We recall that the model addressed for image scalar reconstruction is (2.2) for Gaussian noise and (2.7) for Poisson one.
The optimal scalar hyperparameter corresponding to the n-th patch of the k-th Berkley dataset image is defined as

$$\mu^{k,n}_{GT} \quad \text{with } k \in \{1, \dots, K\},\ n \in \{1, \dots, N\}, \tag{3.3}$$

Each noise type (Gaussian/Poisson) and level (low, high) was applied to the entire image before the patch extraction. Every patch is corrupted by Gaussian or Poisson noise with fixed or different intensities.
Six datasets were thus created divided as follows: 68% for training, 16% for validation, and 16% for testing.

1. Dataset with Gaussian noise of fixed variance $\sigma^2_{AWGN} = 0.01$ and patches size of 64×64 with 37800 patches

2. Dataset with Gaussian noise of fixed variance $\sigma^2_{AWGN} = 0.02$ and patches size of 64×64 with 37800 patches.

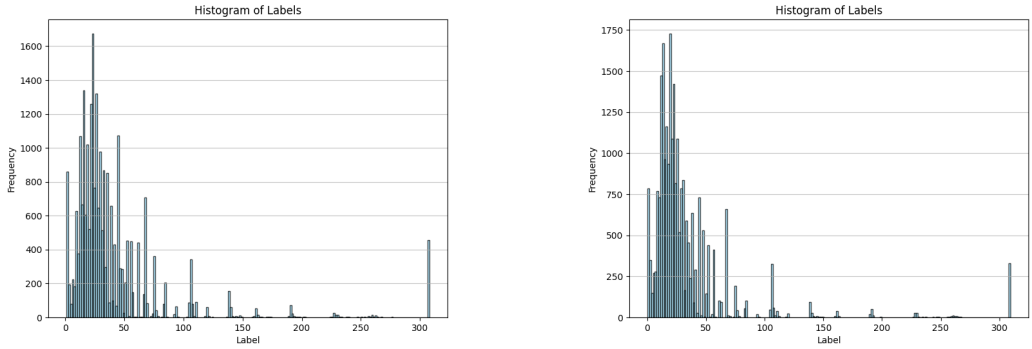
3. Dataset with Gaussian noise of different variance $\sigma^2_{AWGN} = 0.01, \sigma^2_{AWGN} = 0.02$ with patches size of 64×64 with 75600 patches where we have combined both the first and the second dataset.

4. Dataset with Gaussian noise of fixed variance $\sigma^2_{AWGN} = 0.01$ and blur $\sigma_{PSF} = 1.0$ and patches size of 64×64 with with 37800 patches.

5. Dataset with Gaussian noise of fixed variance $\sigma^2_{AWGN} = 0.01$ with patches size of 32×32 with 165300 patches.

6. Dataset with Poisson noise of different intensities with $\alpha = 45, \alpha = 15, \alpha = 5$ with 165300 patches.

Defining as GT_d the d-th dataset created, we analysed the different statistical properties of the labels $\boldsymbol{\mu}^{\mathrm{GT}_d}$ related to the d-th dataset.
Note that the peaks visible in correspondence with the values 0.1 and 308 are due to the fact that those are the lower/upper bounds used in the golden section algorithm.
We intended to not increase this range both because it would have required much more computational time to search in a larger range, but also in order not to have too extreme outliers that could damage the training performance.
We notice from Fig 3.2 that the mean values of the two datasets are quite different, with the former dataset having a higher mean. This outcome is expected, as the second dataset contains higher-intensity Gaussian noise, so to require on average stronger regularization, that is a smaller fidelity parameter μ (i.e. $\mathbb{E}(\boldsymbol{\mu}^{\mathrm{GT}_2}) = 33.28 < \mathbb{E}(\boldsymbol{\mu}^{\mathrm{GT}_1}) = 40.94$).
As shown in the following chapter, learning the distribution of this data distribution was not



(a) $\boldsymbol{\mu}^{\mathrm{GT}_1}$ (b) $\boldsymbol{\mu}^{\mathrm{GT}_2}$

Figure 3.2: Histograms of labels between the dataset with $\sigma^2 = 0.01$ and the one with $\sigma^2 = 0.02$.

difficult, partly because the variance is relatively low, and the high kurtosis reveals clear peaks. Note that in the case of the deblurring dataset, the task renders challenging, due to the high standard deviation ($\sigma(\boldsymbol{\mu}^{\mathrm{GT}_4}) = 53.02$) in the distribution and the low kurtosis ($\mathrm{Kurt}(\boldsymbol{\mu}^{\mathrm{GT}_4}) = 2.25$), indicating that the parameters to be learned are particularly dispersed as illustrated in Fig. 3.3.

While the dataset with Gaussian noise and reduced patches size (Fig. 3.4) shows another highly spread distribution, which is due to the fact that smaller patches can contain an higher variety of scenarios, obtaining an higher variance and lower kurtosis with respect to the 64×64 counterpart of the first dataset (i.e. $\mathrm{Var}(\mu^{\mathrm{GT}_5}) = 53.02 > \mathrm{Var}(\mu^{\mathrm{GT}_1}) = 48.55, \mathrm{Kurt}(\mu^{\mathrm{GT}_5}) =$

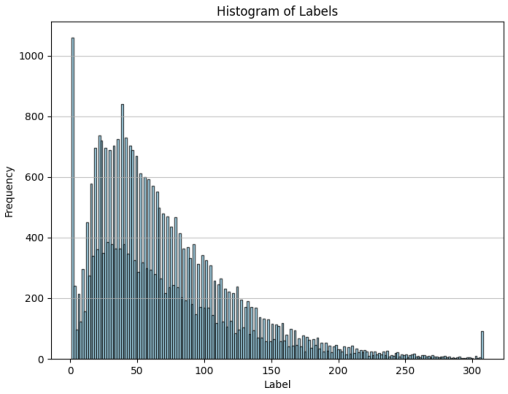


Figure 3.3: Histogram of the dataset labels affected by blur: $\boldsymbol{\mu}^{\mathrm{GT}_4}$.

$10.61 < \mathrm{Kurt}(\mu^{\mathrm{GT}_1}) = 16.44$). These statistics can make it harder for the architecture to differentiate between noise and meaningful textures.

Figure (3.5) shows that the dataset for Poisson noise is concentrated around low values (0.1

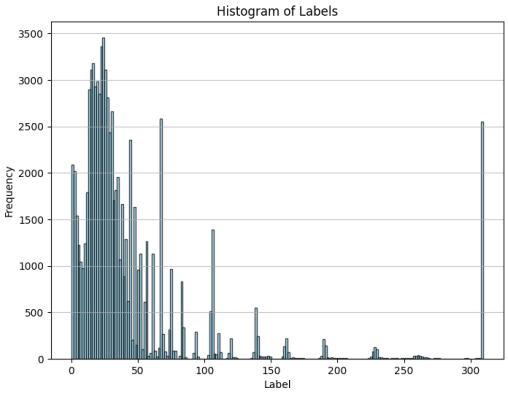


Figure 3.4: Histogram of the dataset labels with reduced patch size $\boldsymbol{\mu}^{\mathrm{GT}_5}$.

- 50). However, the outliers amplify its standard deviation.

We illustrate a table summarising all the main characteristics of the datasets in 3.2.

Dataset	**Mean**	**Std**	**Kurtosis**	**Skewness**
1	40.94	48.55	16.44	3.77
2	33.28	43.02	23.34	4.45
3	37.11	46.03	19.26	4.06
4	70.07	53.02	2.25	1.36
5	45.70	59.49	10.61	3.19
6	23.73	58.93	9.09	3.22

Table 3.2: Summary statistics

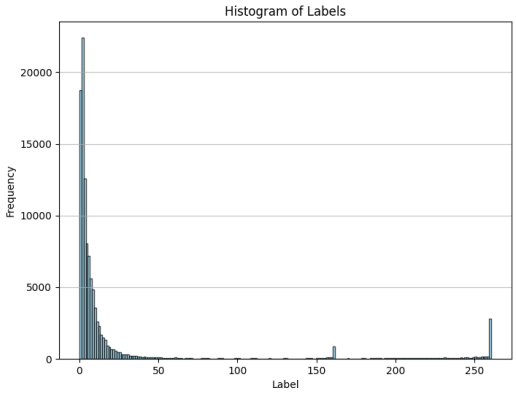


Figure 3.5: Histograms of the labels of the Poisson noise dataset $\boldsymbol{\mu}^{\mathrm{GT}_6}$.

3.3 Deep parameter estimation

We detail in this section the deep learning approach considered to automatically selecting the parameters map of (2.20) to regularize the noisy image. A parameter μ_i has to be found for each pixel within the image.
However, the parameters map may not always be well-defined because the parameters can vary depending on how they are computed. Indeed, computing it patch by patch like in 2.11b can lead to ambiguous results, especially if the patch contains both textured and homogeneous regions.
Obviously, this also depends on the size of the patch itself. The smaller the patch, the greater the likelihood of having statistically similar features throughout the patch.
Our approach to face this difficulties involves selecting a fixed pixel i of one image k, extracting a patch $(\bar{\mathbf{y}}^k)_i \in \mathbb{R}^{64\times 64}$ (or $\mathbb{R}^{32\times 32}$) centered on i, and computing its parameter μ_i based on the surrounding region within the patch itself. This process is repeated for each pixel to capture spatial information across the image.
However, computing these parameters in a SSIM-optimal way, solving the related grid search for each pixel-centered patch is not computationally feasible and would require the ground truth.
To address this issue, a light neural network $f_{\boldsymbol{\theta}}$ with fast inference was trained, to obtain SSIM sub-optimal parameter for each pixel-centered patch, so that

$$f_{\boldsymbol{\theta}}((\bar{\boldsymbol{y}})_i) = (\mu_{DL})_i\,, \tag{3.4}$$

where $(\bar{\boldsymbol{y}})_i$ is the patch centered in the pixel i, $(\mu_{DL})_i$ is the parameter for the pixel i and $\boldsymbol{\theta}$ are the weights of the neural network.
A similar approach was done by Nekhili et al. [2022] and it is shown in their architecture in Fig 3.6.

A significant advantage is that by training our neural network on fixed-size patches, it becomes possible to later apply the network to images of different sizes by extracting sub-regions from them. Although, patches size is an important parameter to tune. Smaller patches are better at capturing the space variance of the image, but they make it more challenging for the neural network to better capture differences between the noise and natural textures.

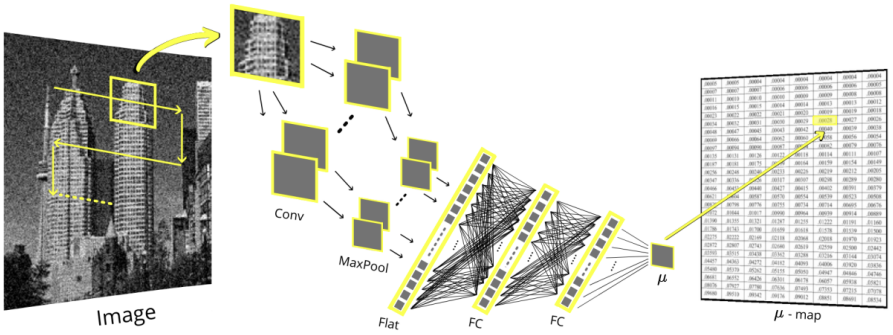


Figure 3.6: CNN architecture for the estimation of the sub-optimal parameter.

3.4 Architecture

In this section, we present the two different architectures that have been proposed based on the size of the patches and the noise analysed.
Both the architectures are very lightweight due to the fact that specifically trained to extract features from small images (i.e. 64×64 or 32×32) and the feature space is not that big to learn. Indeed the learnable parameters of the first architecture are 17823681 ($\sim 10^7$), while the lighter one has 2802817 ($\sim 2 \times 10^6$) learnable parameters.
The first one 3.3 is proposed in the case of patches with size 64×64 and 32×32 in the Gaussian case, while a lighter one 3.4 is proposed for Poisson noise with patches of size 32×32.
The stride and padding parameters of the Conv2D layers are set such that the output size of the image remains the same as the input size. Max pooling layers are used to reduce the image dimensions. While the dropout layers are used for avoiding the overfitting phenomena that we often encountered in our experiments.

3.5 Training

This section describes the training settings used for both architectures before mentioned. Some settings are influenced by the work on regularisation parameters of Afkham et al. [2021], although their experiments were performed on images of size 180×180.
To prevent model overfitting, we reduced the architecture's complexity, but we also implemented the following settings for both architectures:

- Batch_size = 256 to avoid too stochasticity in the gradient direction.
- Learning rate = 10^{-5} to ensure convergence.
- Scheduler of learning rate with an exponential decay of parameter $\gamma = 0.8$.
- Adam optimizer for guaranteeing momentum.
- Weight decay of 10^{-4} for ℓ_2 regularisation.

Layer Type	**Output Shape**	**Kernel / Units**	**Details**
Input Layer	(1, 64, 64)	-	Single-channel input
Conv2D + BatchNorm2D + ReLU	(32, 64, 64)	5×5, 32	Stride 1, Padding 2
Conv2D + BatchNorm2D + ReLU	(64, 64, 64)	5×5, 64	Stride 1, Padding 2
MaxPool2D	(64, 32, 32)	2×2	Stride 2
Conv2D + BatchNorm2D + ReLU	(128, 32, 32)	3×3, 128	Stride 1, Padding 1
MaxPool2D	(128, 16, 16)	2×2	Stride 2
Conv2D + BatchNorm2D + ReLU	(256, 16, 16)	3×3, 256	Stride 1, Padding 1
MaxPool2D	(256, 8, 8)	2×2	Stride 2
Flatten	(16384)	-	Flatten feature maps
Fully Connected + ReLU	(1024)	1024	Fully connected layer
Fully Connected + ReLU	(512)	512	Fully connected layer
Dropout	-	Dropout (0.25)	Regularization
Fully Connected + ReLU	(128)	128	Fully connected layer
Dropout	-	Dropout (0.25)	Regularization
Fully connected	(1)	1	Output layer

Table 3.3: Details of the first architecture.

Layer Type	**Output Shape**	**Kernel / Units**	**Details**
Input Layer	$(1, 32, 32)$	-	Single-channel input
Conv2D + BatchNorm + ReLU	(64,32,32)	$5 \times 5, 64$	Stride=1, Padding=2
MaxPool2D	(64,16,16)	2×2	Stride=2
Conv2D + BatchNorm + ReLU	(128,16,16)	$5 \times 5, 128$	Stride=1, Padding=2
MaxPool2D	(128,8,8)	2×2	Stride=2
Conv2D + BatchNorm + ReLU	(256,8,8)	3×3, 256	Stride=1, Padding=1
MaxPool2D	(256,4,4)	2×2	Stride=2
Conv2D + BatchNorm + ReLU	(512,4,4)	3×3, 512	Stride=1, Padding=1
MaxPool2D	(512,2,2)	2×2	Stride=2
Flatten	2048	-	Flattened output
Fully Connected + ReLU	512	512	Fully connected layer
Dropout	512	-	Dropout (0.25)
Fully Connected + ReLU	128	128	Fully connected layer
Dropout	128	-	Dropout (0.25)
Fully Connected	1	1	Output layer

Table 3.4: Details of the second architecture.

The neural network f_θ is then trained using a Mean Squared Error (MSE) loss (ℓ_2 squared) plus the regularization, so that:

$$\hat{\boldsymbol{\theta}} \in \arg\min_{\boldsymbol{\theta}} \frac{1}{J \cdot N} \sum_{j,n} |f_{\boldsymbol{\theta}}(\mathbf{y}_n^j) - \mu_{GT}^{j,n}|^2 + \lambda \sum_{p=1}^{P} \|\theta\|_2^2 \,, \tag{3.5}$$

with $\lambda = 10^{-4}$ and P the number of learnable parameters of the architecture.
For training purpose, the labels have been normalized using a min-max scaling to map its values to the range [0, 1]:

$$\boldsymbol{\mu}' = \frac{\boldsymbol{\mu} - \min(\boldsymbol{\mu})}{\max(\boldsymbol{\mu}) - \min(\boldsymbol{\mu})}$$

This was done to align the ranges of the labels and features, ensuring compatibility during model training.
The performances of our architectures were assessed using a popular regression metric: the coefficient of determination (R^2) on the test set. Considering N as the number of test patches, we define the error as

$$R^2 := 1 - \frac{\sum_{i=1}^{n}((\mu_{GT})_i - (\mu_{DL})_i}{\sum_{i=1}^{n}((\mu_{GT})_i - \mathbb{E}((\mu_{GT})_i)}. \tag{3.6}$$

The data range of R^2 is $(-\infty,1]$, where 1 indicate a perfect fit and 0 indicate a bad fit, while values below 0 indicates that the model is performing worse than predicting the mean.

An important thing to point out is that training is not computationally expensive, as the light architectures take as input small patches.
The most computationally expensive part of this approach is the creation of the dataset, since for each patch an SSIM-optimal parameter has to be computed. However, once the dataset is created, perform fine-tuning on the architecture and its associated training settings is not very computationally demanding.

3.6 Inference time

In Section 2.1.1 and 2.2.1, we discussed the computational time required to determine the optimal scalar parameter for Gaussian denoising (or deblurring) and Poisson denoising using the golden section algorithm.
Computing the scalar parameter for an image 321×481 requires $\sim 100 - 300$ seconds, while computing the scalar parameter for a patch of size 64×64 (or 32×32) require approximately $\sim 3 - 4$ seconds. However, to compute the scalar parameter for each pixel in a space-variant fashion is unfeasible, because it would require to compute a scalar parameter for $321 \cdot 481 = 154401$ pixels taking approximately 463203 seconds.
One fundamental characteristic of our deep learning approach is that the neural network itself has the property of being very lightweight due to the fact that is specifically trained to extract features from small images (i.e. 64×64 or 32×32) with a feature space not hard to learn.
Thanks to this property of being lightweight our architectures require a short amount of time for performing inference over all the pixels of one image. The network presented in Tab 3.3 requires $\sim$ 277s, while the lighter one presented in Tab 3.4 requires $\sim$ 195s.

Notice that these values does not change based on the noise intensity faced or the presence of blur in the image, unlike classical scalar approaches.

Chapter 4

Results

In this chapter, we report the numerical results obtained for the denoising task for both Poisson and Gaussian noise, injecting noises at varying intensities and carefully analyzing the parameters that influence performances the most. Furthermore, in view of application of the proposed approach to more complex problems, we consider the image deblurring task with known Gaussian blur in the case of Gaussian noise. For each noise distribution, we ran examples taking into account factors such as patch size and the influence of including multiple noise levels in the dataset. We investigated whether these factors improve robustness by enhancing the architecture to better handle unseen noise or improve its performance on noise already seen.
The performances are reported in a table 4.1 with the overall major results. In the table are presented the performances of the regression task using the R^2 formula in (3.6). While for each architecture trained on a given dataset, the mean and standard deviation of SSIM improvement between the scalar and space-variant approaches are evaluated using 30 test images from the Berkeley dataset, corrupted in the same way as the training set.
Since the proposed deep learning approach relies on the prior knowledge of the noise distribution, we also developed a classification network to determine the noise type (Poisson or Gaussian) as described in the first section.

Dataset	R^2	**Level Tested**	**Mean**	**Std**
1	0.6240	$\sigma^2_{AWGN} = 0.01$	0.002	0.028
2	0.4939	$\sigma^2_{AWGN} = 0.02$	-0.006	0.036
3	0.6135	$\sigma^2_{AWGN} = 0.02$	0.0004	0.031
3	0.6135	$\sigma^2_{AWGN} = 0.015$	0.006	0.027
4	0.6124	$\sigma^2_{AWGN} = 0.01$	-0.0007	0.042
5	0.6455	$\sigma^2_{AWGN} = 0.01, \sigma^2_{PSF} = 1.0$	0.005	0.018
6	0.3936	$\alpha = 45$	-0.0059	0.148

Table 4.1: Summary of results for six datasets, including R^2 values, tested noise levels, and SSIM improvement metrics.

4.1 Classification task

Given the statistical differences of the noise observed in both the additive Gaussian and in the signal-dependent Poisson case, a natural question on whether it's possible to discriminate between them (binary classification problem) using a simple classifier arises. This could favour ad-hoc processing with the appropriate data term, see equations (2.21) or (2.22).
The dataset for this task was created by extracting patches of size 64×64. Within the dataset, both types of noise, Gaussian and Poisson, were simulated with three different intensities for each type of noise (i.e. $\sigma^2_{AWGN} = 0.01, \sigma^2_{AWGN} = 0.02, \sigma^2_{AWGN} = 0.03, \alpha = 45, \alpha = 15, \alpha = 5$). Gaussian noise patches were labelled as zero, while the Poisson ones were labelled as one.
We used the architecture presented in Tab 3.3 as feature extractor for the image patches and the output shape of the last multi-layer perceptron was setted as 2, in order to output two probabilities i.e. Gaussian or Poisson.
At test time the accuracy obtained is of **92%** leading to good results at inference time with a stable validation loss presented in Fig 4.1.
The idea is to extract a fixed number of patches from an image and check if the network can

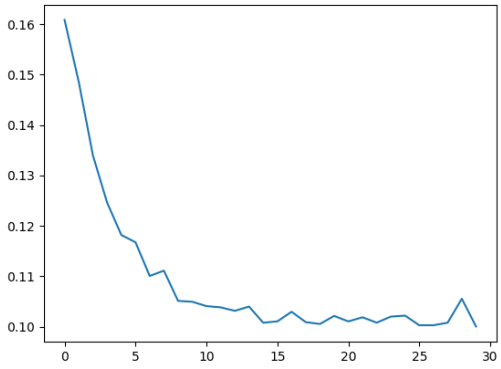
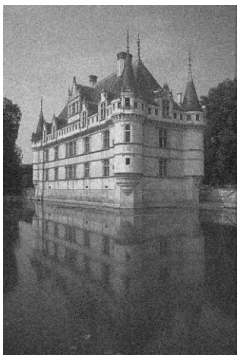


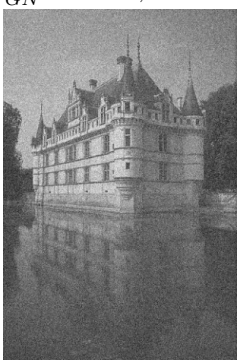
Figure 4.1: Validation loss for classification task.

correctly classify, with a good accuracy, as Gaussian(or Poisson) the entire image is classified as Gaussian(or Poisson).
When tested on the castle affected by different intensity of Gaussian noise we obtain good accuracy even when the noise is almost imperceptible (i.e. $\sigma^2_{AWGN} = 0.001$) like in Fig 4.2. While we obtain slightly worst results in the Poisson case (Fig 4.3), especially in the case of imperceptible noise (i.e. $\alpha = 65$), but still appreciable.

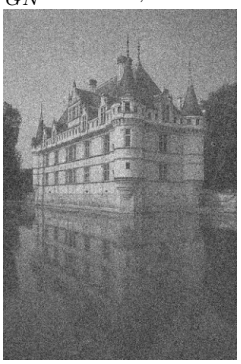
(a) $\sigma^2_{AWGN} = 0.001$, accuracy : 72%



(b) $\sigma^2_{AWGN} = 0.005$, accuracy : 89%

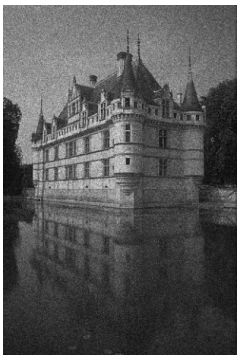


(c) $\sigma^2_{AWGN} = 0.01$, accuracy : 100%

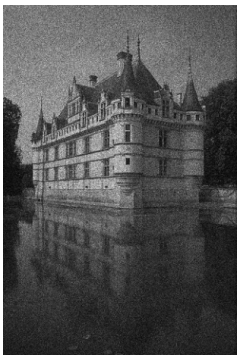


(d) $\sigma^2_{AWGN} = 0.02$, accuracy : 100%

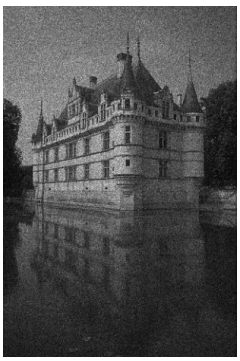
Figure 4.2: Classification of Gaussian noise based on different intensities.



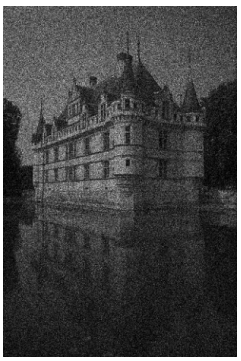
(a) $\alpha = 65$, accuracy : 60%



(b) $\alpha = 55$, accuracy : 69%



(c) $\alpha = 45$, accuracy : 74%



(d) $\alpha = 15$, accuracy : 100%

Figure 4.3: Classification of Poisson noise based on different intensities.

4.2 Gaussian denoising

In this section, we experimented with different datasets for Gaussian denoising. We started by addressing a single noise level, ranging from low to medium, and finally we expanded the approach to handle multiple noise levels. Additionally, we compared the space-variant reconstruction performances with respect to the scalar one using patch sizes of either 64×64 or 32×32.

4.2.1 Low noise ($\sigma^2_{AWGN} = 0.01$)

We started with a dataset (1th one) containing patches size of 64×64 and Gaussian noise with $\sigma^2_{AWGN} = 0.01$.
The R^2 value obtained is 0.6240 and the training phase was stopped at 25 epochs, as the validation loss began showing oscillatory behaviour and the train loss was almost flat after only a few epochs. We report these convergence plots in Fig 4.4.

(a) Training loss (b) Validation loss

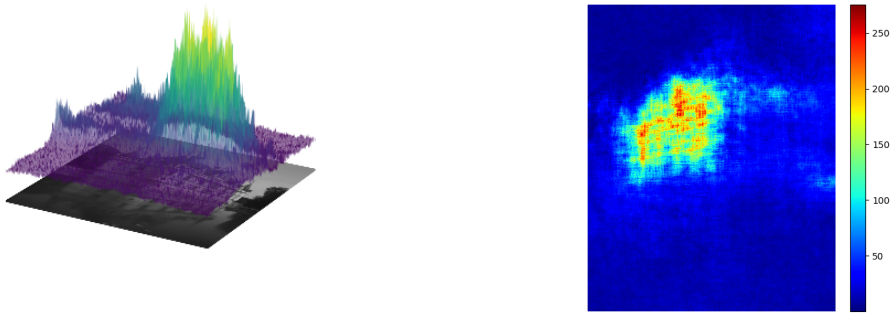
Figure 4.4: Losses for denoising task with $\sigma^2_{AWGN} = 0.01$.

Testing the architecture on a test image we achieved in Fig 4.5 remarkable results using the space-variant approach, improving the SSIM value compared to the scalar method and preserving textured regions, see, for instance, the fine-detailed areas of the castle from being over-smoothed as seen with the global approach.

(a) $\boldsymbol{y}$, SSIM = 0.38 (b) $\tilde{\boldsymbol{x}}$, SSIM = 0.75 - $\tilde{\boldsymbol{x}}^{SV}$, SSIM = 0.78

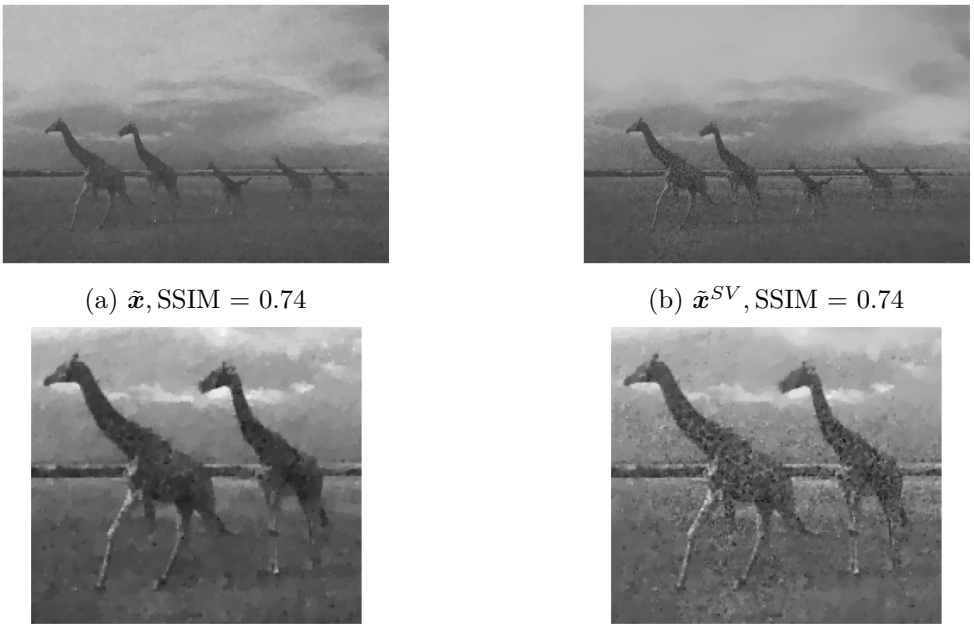
Figure 4.5: Noisy image with its scalar and space-variant reconstruction for $\sigma^2_{AWGN} = 0.01$

The estimated parameter map behaves as expected, i.e. it applies less regularization in highly textured areas and smooths out with stronger strength homogeneous (piece-wise constant) areas. This can be observed in Fig. 4.6, where the castle is practically segmented upon estimation of the parameter map and subject to significantly lower regularisation i.e. higher value of the parameters in that region.
We tested the architecture on 30 test images and the results found show a very small average SSIM improvement of 0.0024 against the 0.03 provided in castle example. This is mainly because our approach achieves good SSIM improvements on some images but may obtain SSIM decrease on others. Indeed, the standard deviation of the SSIM improvements is pretty high ($\sigma = 0.028$) as reported in Tab 4.1.
However, it is worth highlighting that even when the SSIM value remains similar to the one found in global approach, the visual improvements are still highly appreciated, as in Fig 4.7.



(a) 3D $\boldsymbol{\mu} - map$ (b) Colorbar of $\boldsymbol{\mu} - map$

Figure 4.6: 3D graph-2D colorbar of the parameters map for $\sigma^2_{AWGN} = 0.01$ Gaussian denoising.

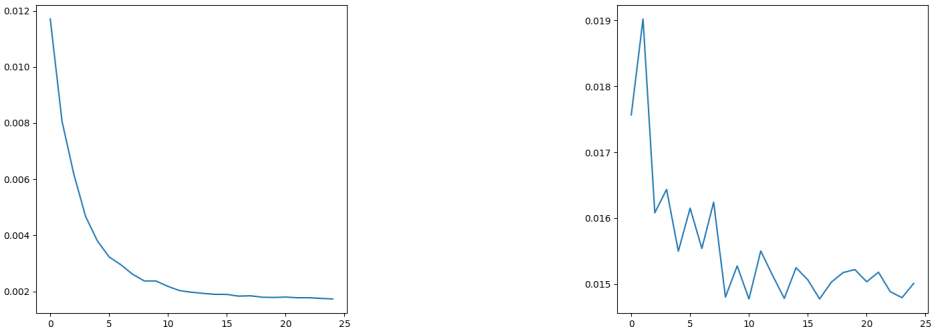


(a) $\tilde{\boldsymbol{x}}$, SSIM = 0.74 (b) $\tilde{\boldsymbol{x}}^{SV}$, SSIM = 0.74

Figure 4.7: Images with same SSIM values, but different preserved texture.

4.2.2 Medium noise ($\sigma^2_{AWGN} = 0.02$)

In the next phase, we consider a dataset (2th) corrupted by higher Gaussian noise with a variance $\sigma^2_{AWGN} = 0.02$.
To test this, we considered a new image dataset with patches of the same size but with different realizations of Gaussian noise at this new higher intensity. Having a higher noise leads of a decreasing in optimal parameters value (indeed $\mathbb{E}(\boldsymbol{\mu}_{GT_2}) = 39.10 < \mathbb{E}(\boldsymbol{\mu}_{GT_1}) = 48.25$) since more regularization is therein required.
Convergence is achieved in approximately 25 epochs, as showed in Fig 4.8. The $R^2 = 0.4939$ value in this case is way lower with respect to the low level noise case ($R^2 = 0.6240$), suggesting that higher noise indicates a more challenging regression task and slightly lower performances as showed in Tab 4.1.



(a) Training loss (b) Validation loss

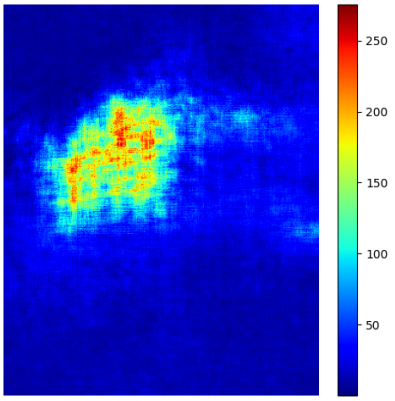
Figure 4.8: Losses for denoising task with $\sigma^2_{AWGN} = 0.02$.

However, for consistency, we still used the image of the castle in Figure 4.5 for inference, observing a good improvement with respect to the scalar approach. Despite the SSIM improvement is lower than the previous case, the visual effects remain remarkable as shown in Fig 4.9.

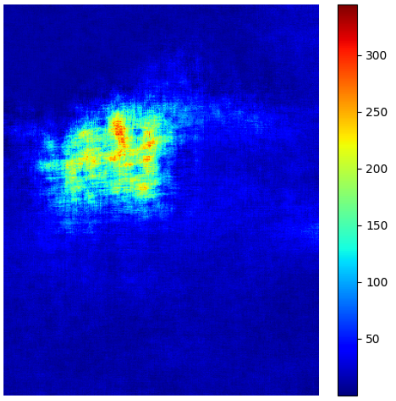
(a) $\boldsymbol{y}$, SSIM = 0.30 (b) $\tilde{\boldsymbol{x}}$, SSIM = 0.69 - $\tilde{\boldsymbol{x}}^{SV}$, SSIM = 0.70

Figure 4.9: Noisy image with its scalar and space-variant reconstruction for $\sigma^2_{AWGN} = 0.02$.

With a 2D colorbar map (Fig 4.10) we show how the peaks around the castle remain, although less pronounced (i.e. more regularisation) in the medium noise level, for both tests.



(a) 2D colorbar of $\boldsymbol{\mu} - map$

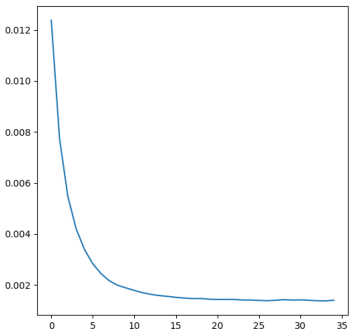


(b) 2D colorbar of $\boldsymbol{\mu} - map$

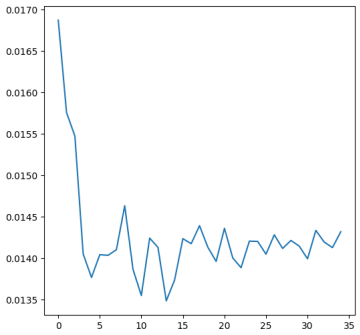
Figure 4.10: 2D colorbar for the Gaussian denoising tested on $\sigma^2_{AWGN} = 0.01, \sigma^2_{AWGN} = 0.02$.

4.2.3 Enhancing robustness

Using the 3th dataset, we showed how to improve the robustness of the deep learning approach, training a neural network on two noise levels and then testing it on noise levels not provided during training, analysing the performances obtained in these scenarios. The idea was to combine the two previously used datasets and test the model on unseen noise i.e. $\sigma^2_{AWGN} = 0.015$. In addition, we found that the network trained on mixed noise (0.01 and 0.02) not only performed well on unseen noise but also achieved better results on images corrupted by noise whose intensity was seen during training as it has been shown in the results table 4.1. This can also be observed in Fig 4.12 where a small improvement of (~ 0.005) is obtained on the noisy image corrupted by one of the two training noise levels illustrated before in Fig 4.9.
Even in this case, only 25 epochs were sufficient, as seen in Fig (4.11), as convergence was achieved quickly and the $R^2 = 0.6135$ value obtained is way higher with respect to the medium level noise intensity.

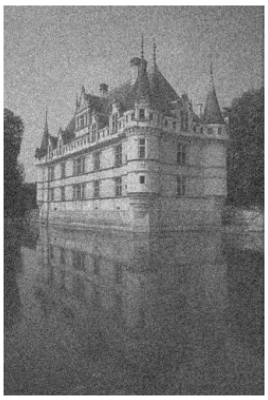


(a) Training loss

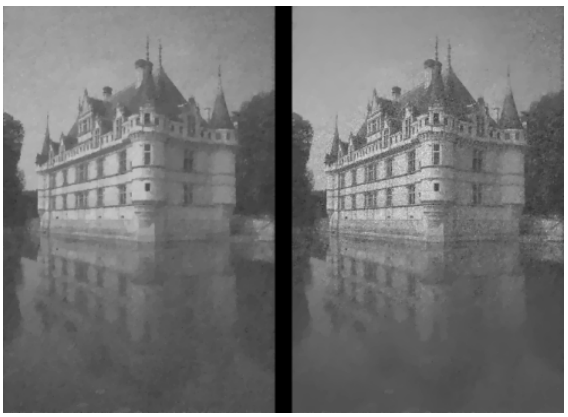


(b) Validation loss

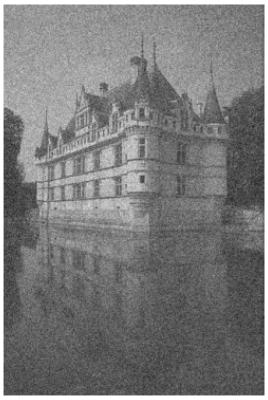
Figure 4.11: Losses for denoising task with $\sigma^2_{AWGN} = 0.01$,0.02.



(a) $\boldsymbol{y}$, SSIM = 0.34



(b) $\tilde{\boldsymbol{x}}$, SSIM = 0.71 - $\tilde{\boldsymbol{x}}^{SV}$, SSIM = 0.73



(c) $\boldsymbol{y}$, SSIM = 0.30

(d) $\tilde{\boldsymbol{x}}$, SSIM = 0.69 - $\tilde{\boldsymbol{x}}^{SV}$, SSIM = 0.71

Figure 4.12: Left: noisy images with different Gaussian noise levels. Right: scalar and space variant reconstructions for $\sigma^2_{AWGN} = 0.015$ and $\sigma^2_{AWGN} = 0.02$ using the architecture trained on mixed noises.

4.2.4 Ablation study on patch sizes

Patch size is an important parameter to tune: small patches can capture more space-variance and can better adapt to detect complex structure in the whole image than larger one. However it may be more difficult for the architecture to learn a successful regression task in this case as the extraction of meaningful features from smaller images is not trivial.
To test this, we considered a dataset (4th) with patches size of 32×32 and fixed noise variance $\sigma^2_{AWGN} = 0.01$. We repeated the training using the architecture from all previous cases (i.e. the more complex one in Tab 3.3), but this time a max pooling layer was removed to ensure that the output shape after the flattening operation was still 16384.
Examining this dataset to the first, where the variance remains the same but the patch size decreases: the R^2 value of this dataset is slightly lower to the one exhibited in the first dataset. However, it is a good value considering that learning from smaller patches is indeed a more challenging task.
Although the quantitative SSIM results are approximately similar for both the 64x64 and 32x32 patch sizes as shown in Tab 4.1, the visual results and adaptivity observable in the regularisation parameters map are far more appreciable in the case of smaller patches.
Indeed, the visual improvements in Figure 4.13 are appreciable as the fur (and wings) can be seen better.

Furthermore, using smaller patches allow to adapt to more complex shapes and also succeed

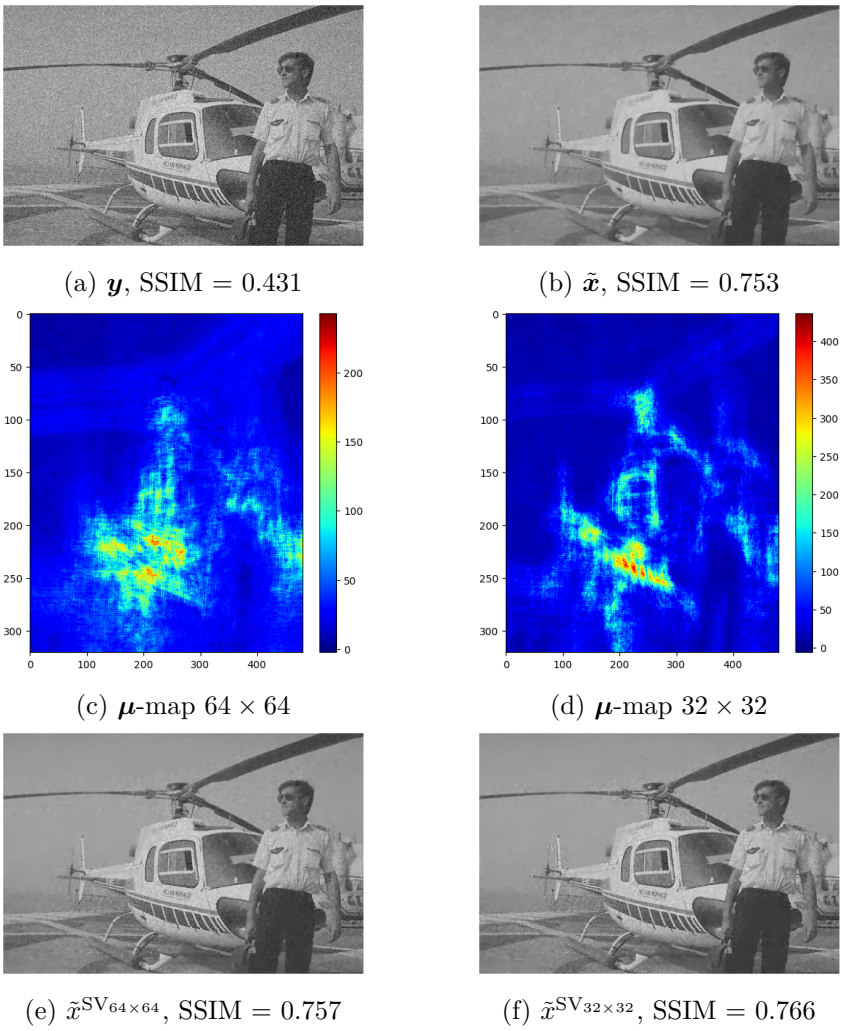
(a) $\boldsymbol{y}$, SSIM = 0.30

(b) $\tilde{\boldsymbol{x}}^{\mathrm{SV}_{64\times64}}$, SSIM = 0.882

(c) $\tilde{\boldsymbol{x}}^{\mathrm{SV}_{32\times32}}$, SSIM = 0.876

Figure 4.13: Space-variant reconstruction by architectures trained on 64×64 and 32×32.

to regularise regions where thinner objects are embedded in homogeneous regions, which otherwise would create noise clouds due to the patch size being too large compared to the object itself. A clear example is the helicopter's blade in Fig 4.14.



(a) $\boldsymbol{y}$, SSIM = 0.431

(b) $\tilde{\boldsymbol{x}}$, SSIM = 0.753

(c) $\boldsymbol{\mu}$-map 64×64

(d) $\boldsymbol{\mu}$-map 32×32

(e) $\tilde{x}^{\mathrm{SV}_{64\times64}}$, SSIM = 0.757

(f) $\tilde{x}^{\mathrm{SV}_{32\times32}}$, SSIM = 0.766

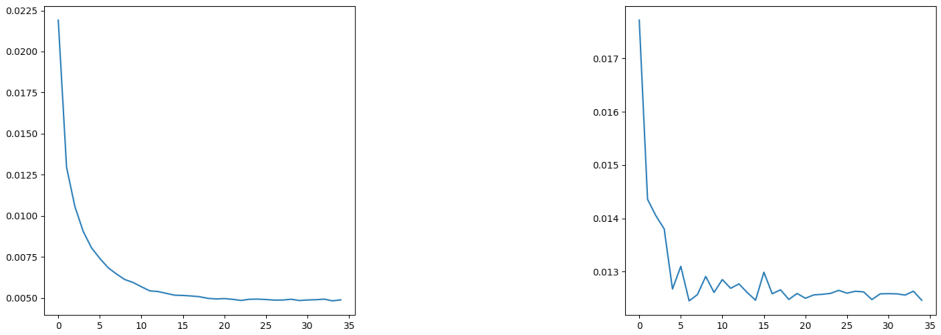
Figure 4.14: Better reconstruction of fine-grained details in the denoising case with $\sigma^2_{AWGN} = 0.01$.

4.3 Gaussian deblurring

We now consider the case where the forward operator is a matrix defining the convolution operation between the image and a Gaussian kernel. For this new task we have created a dataset (5th) with patches sizes of 64×64, where we applied a Gaussian blur using the FFT algorithm with a variance of $\sigma^2_{PSF} = 1.0$ and then we added Gaussian noise of variance $\sigma^2_{AWGN} = 0.01$.

Applying a Gaussian kernel tends to smooth the image, as the convolution tends to average out the values present in the part affected by the filter by attenuating the high frequencies, i.e. by reducing the intensity shifts. This leads to the smoothing of edges and a general reduction of highs and lows, thus reducing the overall image variance. Furthermore, as seen in the previous chapter, the distribution of parameters in the deblurring case is very sparse, with a very high mean value.

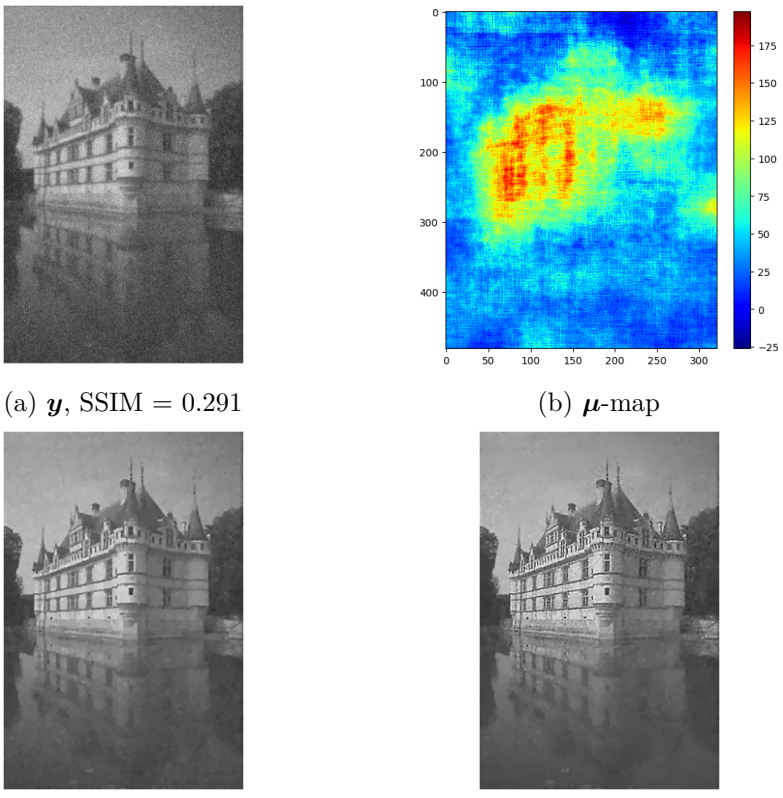
Despite the Gaussian convolution tend to reducing the pixel intensity range, the achieved R^2 value is the best among all the regression tasks and it is also worth mentioning that the training in Fig 4.15 was more stable than in the denoising case illustrated in Fig 4.4 and 4.8.



(a) Training loss (b) Validation loss

Figure 4.15: Losses for deblurring task with $\sigma^2_{AWGN} = 0.01, \sigma^2_{PSF} = 1.0$.

We observe from Fig 4.16 that, the parameters map looks smoother in comparison to the ones computed for the denoising task, probably due to the blurring and averaging effects introduced by the Gaussian filter. However this smoothed map do not interfere with a good reconstruction of the image.



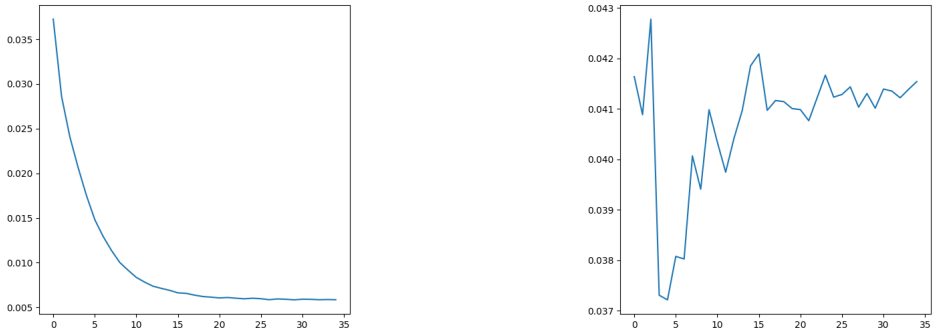
(a) $\boldsymbol{y}$, SSIM = 0.291 (b) $\boldsymbol{\mu}$-map

(c) $\tilde{\boldsymbol{x}}$, SSIM = 0.758 (d) $\tilde{\boldsymbol{x}}^{\mathrm{SV}}$, SSIM = 0.777

Figure 4.16: Blurred image with its global and space-variant reconstruction for $\sigma^2_{AWGN} = 0.01, \sigma^2_{PSF} = 1.0$.

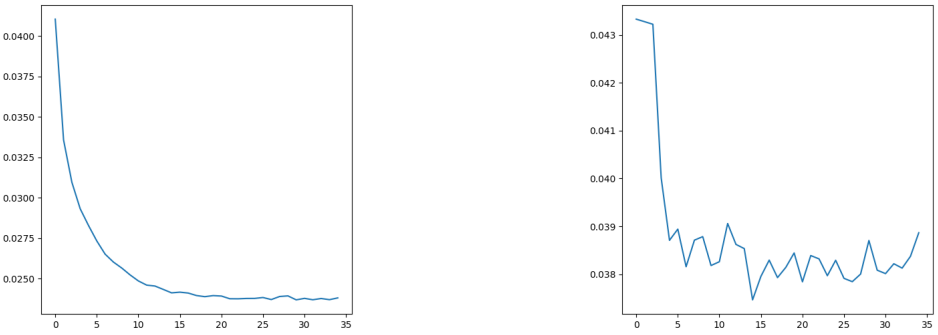
4.4 Poisson denoising

Based on the previous experiments, we now stick with the 32 x 32 patches for performing denoising on a different, non-additive, Poisson noise model.
The Poisson dataset (6th) was created with 32×32 patches and noise of three different intensities: $\alpha_1 = 45, \alpha_2 = 15, \alpha_3 = 5$, where the lower the alpha parameter is, the more intense the noise.
In this new context, the complex architecture adapted for patch size of 32×32 (i.e. the architecture presented in Tab 3.3 with the removal of one max pooling layer) cause issues related to overfitting as displayed in Fig 4.17 where an increasing trend in the validation loss appears.
One of the most effective methods to reduce overfitting is to simplify the architecture. On one side, this simple idea is such that the architecture learns only the general features of the task, and not features of a particular dataset.
We used the lighter architecture presented in Tab 3.4 that obtain a more stable training, as shown by the validation loss illustrated in Fig 4.18, with respect to the complex one.



(a) Training loss (b) Validation loss

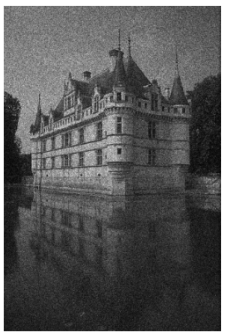
Figure 4.17: Losses(big architecture) for denoising task in Poisson case for $\alpha_1 = 45, \alpha_2 = 15, \alpha_3 = 5$.



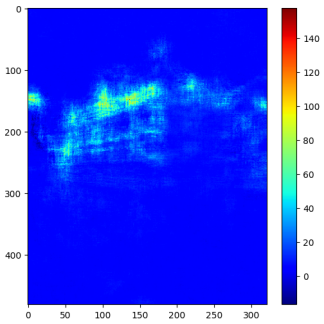
(a) Training loss (b) Validation loss

Figure 4.18: Losses(small architecture) for denoising task in Poisson case for $\alpha_1 = 45, \alpha_2 = 15, \alpha_3 = 5$.

It is worth mentioning that the regression task for Poisson denoising achieve the worst performances across all the datasets tested, obtaining a small value of $R^2 = 0.3936$. It is likely that this error arises from cases where the Poisson noise is more intense (i.e. $\alpha = 5$).
Furthermore, Poisson noise is signal-dependent, hence it may poses greater challenges for regression task and inference. However, when tested on images of low intensity Poisson noise (e.g., higher α parameter), improvements could still be observed.
Testing, for consistency, the usual castle image, our approach performs well on low intensity Poisson noise (i.e. $\alpha = 45$) as it has been exhibited in Fig 4.19, but it starts to have some decrease in the SSIM value when tested on higher intensity noise like in Fig 4.20, even thought the visual effect remains acceptable.



(a) $\boldsymbol{y}$, SSIM = 0.390

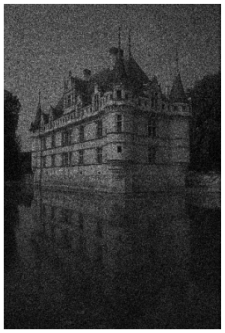


(b) $\boldsymbol{\mu}$-map

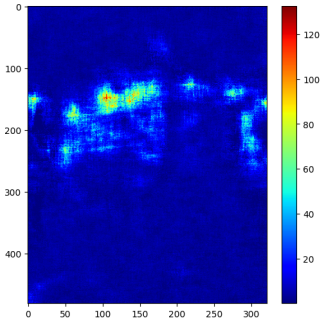
(c) $\tilde{\boldsymbol{x}}$, SSIM = 0.668

(d) $\tilde{\boldsymbol{x}}^{\mathrm{SV}}$, SSIM = 0.700

Figure 4.19: (Poisson) noisy image with its global and space-variant reconstruction for $\alpha = 45$.

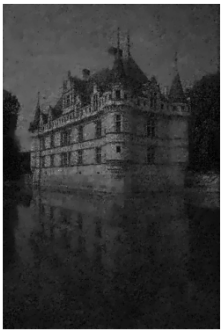


(a) $\boldsymbol{y}$, SSIM = 0.232



(b) $\boldsymbol{\mu}$-map

(c) $\tilde{\boldsymbol{x}}$, SSIM = 0.492



(d) $\tilde{\boldsymbol{x}}^{\mathrm{SV}}$, SSIM = 0.475

Figure 4.20: (Poisson) noisy image with its global and space-variant reconstruction for $\alpha = 15$.

Chapter 5

Conclusions and outlook

In this work, we addressed the task of estimating adaptive hyper-parameters in a variational model designed for reconstructing natural images corrupted by Gaussian noise, Poisson noise and, in some cases, Gaussian blur. In particular, we developed a space-variant model using a deep-learning approach for Total-variation(TV) regularisation combined with a suitable data term tailored to the noise model considered. Moving from a scalar to a pixel-dependent estimation adapts more effectively to the heterogeneous image details, thus improving overall reconstruction quality.
For the estimation of the parameter map, we divided a dataset of natural images into patches and developed a supervised training procedure of scalar regularisation parameters to be placed at the centre of sliding window running through the image by means of a (light) deep neural network. Upon this training, at inference time the trained network is thus able to infer a regularisation map for the image at hand.
The space-variant approach proved slightly better results with respect to the classical scalar approach in term of SSIM metric, with meaningful and visible improvements from a visual perspective, preserving textured regions and smoothing out homogeneous (piece-wise constant) areas. Furthermore, the computational cost of the computation of the space-variant map is extremely fast, being the evaluation of a trained network in correspondence of the given image considered .
Training was performed efficiently by using patch datasets with different noise statistics and sizes. Due to the light architecture employed and the limited size of the training dataset, training was computationally effective and further benefited from the use of accelerated first-order methods and advanced backtracking strategies for the solution of the TV-regularised problems.
Testing the network performances under a variety of scenarios provided meaningful insights regarding the factors that contribute the most to a successful image reconstruction.

- Smaller patches lead to better segmentation of the parameters map and subsequently better visual results. A slight SSIM improvement has been noted on some images with respect to the scalar case. However, from a visual perspective, the results are significantly improved in textured areas, thus allowing better detail preservation.

- Including multiple noise levels in the dataset (from low to high) provide better performances on unseen noise and even already seen noise.

As an outlook for future research we mention:

- Architectures that perform better on the regression task used to train the network upon the use of scalar training should be explored more. Unlike what we have done in our approach, the architecture should be adapted based on the type of noise and the patch size used.
- Reducing the patch size may lead to better segmentation of the parameters map and improved local adaptability. Different architectures should thus be explored to better handle this new patch size.