

Proximal Neural Networks: Wedding Variational Methods and Artificial Intelligence

VI – Conclusion and toolbox presentation

Audrey REPETTI[†], Nelly PUSTELNIK[◇], Jean-Christophe PESQUET^{*}

^{*} CentraleSupélec, Université Paris-Saclay, Gif-sur-Yvettes, France

[◇] CNRS, ENS Lyon, Lyon, France

[†] Heriot-Watt University & Maxwell Institute for Mathematical Sciences, Edinburgh, UK

TUTORIAL – EUSIPCO 2025 – Palermo, Italy

Unified framework

Inference framework: feed-forward NN

$$\begin{aligned}(\forall \mathbf{x}^{[0]} \in \mathbb{R}^{N_0}) \quad \mathbf{x}^{[K]} &= \mathfrak{L}_{\Theta}^K(\mathbf{x}^{[0]}) \\ &= \mathfrak{T}_{\Theta_K} \circ \dots \circ \mathfrak{T}_{\Theta_1}(\mathbf{x}^{[0]}),\end{aligned}$$

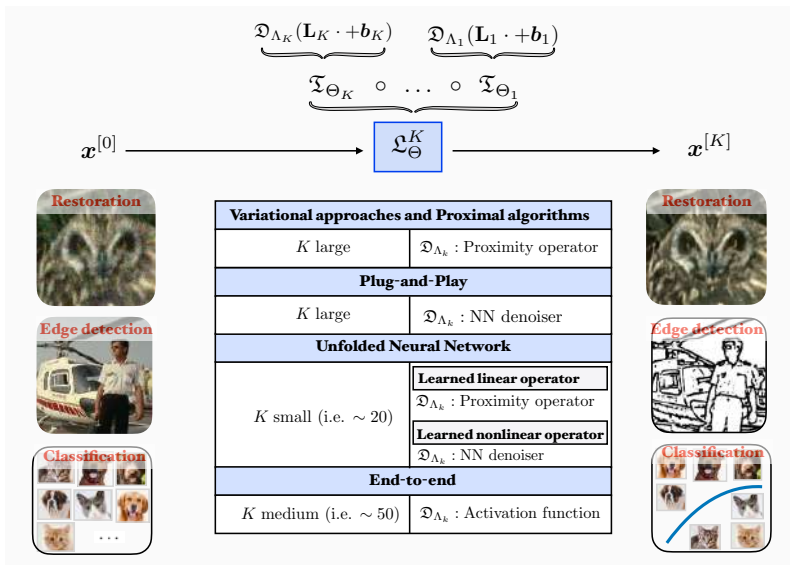
Layer/iteration

$$\mathfrak{T}_{\Theta_k} : \mathbb{R}^{N_{k-1}} \rightarrow \mathbb{R}^{N_k} : \mathbf{x} \mapsto \mathfrak{D}_{\Lambda_k}(\mathbf{L}_k \mathbf{x} + \mathbf{b}_k),$$

- ▶ $\mathbf{L}_k : \mathbb{R}^{N_{k-1}} \rightarrow \mathbb{R}^{N_k}$: linear operator,
- ▶ $\mathbf{b}_k \in \mathbb{R}^{N_k}$: shift parameter,
- ▶ $\mathfrak{D}_{\Lambda_k} : \mathbb{R}^{N_k} \rightarrow \mathbb{R}^{N_k}$: nonlinear operator parametrized by Λ_k .

Parameters: $\Theta = \cup_{k=1}^K \Theta_k$ with $\Theta_k = \{\Lambda_k, \mathbf{L}_k, \mathbf{b}_k\}$.

Unified framework: Unfolded neural networks



Challenges for the next years

THEORETICAL CHALLENGES:

- Interpretation of output of (unfolded) neural networks
- Develop mathematical framework to better assess robustness of (unfolded) neural networks

COMPUTATIONAL CHALLENGES:

- Boost expressivity of PnP and unfolded methods
- Further explore real applications

SOCIETAL CHALLENGES:

- Convince end-users that model-informed deep learning methods such as PnP and unfolded networks are reliable for decision-making processes
- Develop effective quantification measures for environmental impact of data-driven methods
 - ↪ Reduce environmental impact by adoption of frugal learning strategies

Soon(ish) available...



From Iterative Methods to Model-Informed Architectures for Data Science.

A. Repetti, N. Pustelnik, J.-C. Pesquet.

To be submitted

↪ *Review article from proximal methods to PnP and unfolded approaches*

PYTHON TOOLBOX

PLAYING WITH INVERSE IMAGING PROBLEMS

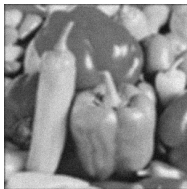
Forward model

FORWARD MODEL: $z = \mathcal{D}(\mathbf{A}\bar{x})$

- $\bar{x} \in \mathcal{H}$ original unknown image
- $z \in \mathcal{G}$ degraded measurements
- $\mathbf{A}: \mathcal{H} \rightarrow \mathcal{G}$ corresponds to the linear measurement operator
- $\mathcal{D}: \mathcal{G} \rightarrow \mathcal{G}$ models the degradation noise

OBJECTIVE: Find an estimate $\hat{x} \in \mathcal{H}$ of the original image \bar{x} from the measurements z

EXAMPLE: Image restoration (e.g., deblurring)



Observation



Estimate

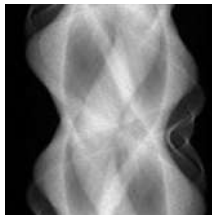
Forward model

FORWARD MODEL: $z = \mathcal{D}(\mathbf{A}\bar{x})$

- $\bar{x} \in \mathcal{H}$ original unknown image
- $z \in \mathcal{G}$ degraded measurements
- $\mathbf{A}: \mathcal{H} \rightarrow \mathcal{G}$ corresponds to the linear measurement operator
- $\mathcal{D}: \mathcal{G} \rightarrow \mathcal{G}$ models the degradation noise

OBJECTIVE: Find an estimate $\hat{x} \in \mathcal{H}$ of the original image \bar{x} from the measurements z

EXAMPLE: Medical imaging (CT)



Observation



Estimate

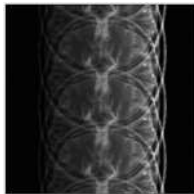
Forward model

FORWARD MODEL: $z = \mathcal{D}(\mathbf{A}\bar{x})$

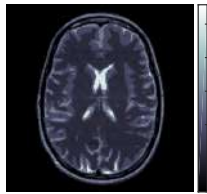
- $\bar{x} \in \mathcal{H}$ original unknown image
- $z \in \mathcal{G}$ degraded measurements
- $\mathbf{A}: \mathcal{H} \rightarrow \mathcal{G}$ corresponds to the linear measurement operator
- $\mathcal{D}: \mathcal{G} \rightarrow \mathcal{G}$ models the degradation noise

OBJECTIVE: Find an estimate $\hat{x} \in \mathcal{H}$ of the original image \bar{x} from the measurements z

EXAMPLE: Magnetic resonance imaging in medicine



Observation



Estimate

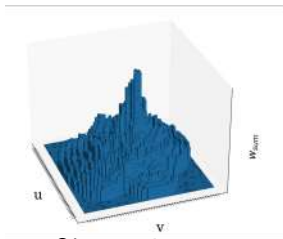
Forward model

FORWARD MODEL: $z = \mathcal{D}(\mathbf{A}\bar{x})$

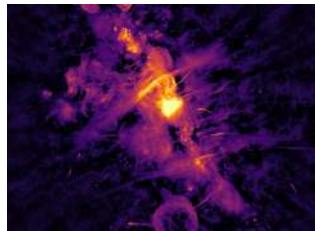
- $\bar{x} \in \mathcal{H}$ original unknown image
- $z \in \mathcal{G}$ degraded measurements
- $\mathbf{A}: \mathcal{H} \rightarrow \mathcal{G}$ corresponds to the linear measurement operator
- $\mathcal{D}: \mathcal{G} \rightarrow \mathcal{G}$ models the degradation noise

OBJECTIVE: Find an estimate $\hat{x} \in \mathcal{H}$ of the original image \bar{x} from the measurements z

EXAMPLE: Radio-interferometric imaging in astronomy



Observation



Estimate

Forward model: Examples for measurement operator \mathbf{A}

- **Deconvolution**

- Most common imaging model encountered in the literature, also known as deblurring
- \mathbf{A} associated with a 2D or 3D convolution (or blur) kernel
- For example to **model motion** between the scene and the camera, for **defocusing** of an optical imaging system, or to **model atmospheric turbulence** (e.g., in astronomical or satellite imaging)

Forward model: Examples for measurement operator \mathbf{A}

- **Deconvolution**
- **Subsampling/inpainting**
 - \mathbf{A} corresponds to a *mask* operator, only selecting visible pixels
 - Used to **model missing information**, for example in the context of low-resolution acquisition (i.e., *super-resolution*), or from an occultation process (i.e., *inpainting*).

Forward model: Examples for measurement operator \mathbf{A}

- **Deconvolution**
- **Subsampling/inpainting**
- **Fourier sampling**
 - \mathbf{A} can be decomposed into two linear operators: the discrete Fourier transform (i.e., 2D FFT), and the subsampling operator
 - ↪ In a realistic setting, the Fourier transform should act in a continuous space (i.e., using *non-uniform FFT*)
 - Encountered for instance in medicine for magnetic resonance imaging, and in astronomy for radio-interferometric imaging

Forward model: Examples for measurement operator \mathbf{A}

- **Deconvolution**
- **Subsampling/inpainting**
- **Fourier sampling**
- **Radon transform**
 - \mathbf{A} produces a 2D (or 3D) sinogram
 - Usually used to approximate tomography projection operators as encountered for Positron Emission Tomography (PET) or Computed Tomography (CT)
- **etc.**

Forward model: Link between degradation \mathcal{D} and data fidelity h_z

FORWARD MODEL: $z = \mathcal{D}(\mathbf{A}\bar{x})$

VARIATIONAL FORMULATION: Define the estimate \hat{x} as $\mathbf{0} \in \partial h_z(\hat{x}) + \lambda \partial g(\hat{x})$

- **Additive white Gaussian noise (AWGN)**

- Most common type of noise encountered in practice
- Model boils down to $z = \mathbf{A}\bar{x} + \varepsilon$ where $\varepsilon \in \mathcal{G}$ is a realization of an independent identically distributed random Gaussian variable with zero mean and standard deviation $\sigma > 0$ (or diagonal covariance Σ)
- $h_z(x) = \frac{1}{2\sigma^2} \|\mathbf{A}x - z\|^2$

Forward model: Link between degradation \mathcal{D} and data fidelity h_z

FORWARD MODEL: $z = \mathcal{D}(\mathbf{A}\bar{x})$

VARIATIONAL FORMULATION: Define the estimate \hat{x} as $\mathbf{0} \in \partial h_z(\hat{x}) + \lambda \partial g(\hat{x})$

- Additive white Gaussian noise (AWGN)
- Coloured Gaussian noise
 - More general version of AWGN where the the covariance Σ of the noise is not diagonal
 - $h_z(x) = \frac{1}{2} \|\mathbf{A}x - z\|_{\Sigma^{-1}}^2$

Forward model: Link between degradation \mathcal{D} and data fidelity h_z

FORWARD MODEL: $z = \mathcal{D}(\mathbf{A}\bar{x})$

VARIATIONAL FORMULATION: Define the estimate \hat{x} as $\mathbf{0} \in \partial h_z(\hat{x}) + \lambda \partial g(\hat{x})$

- **Additive white Gaussian noise (AWGN)**
- **Coloured Gaussian noise**
- **Poisson noise**
 - Often used to model noise in low-photon-count imaging techniques
 - ↪ Poisson distribution is a counting procedure that can express the number of photons received by the sensor in a given time interval
 - $h_z(x) = \sum_m ([\mathbf{A}x]_m - z_m \log([\mathbf{A}x]_m))$

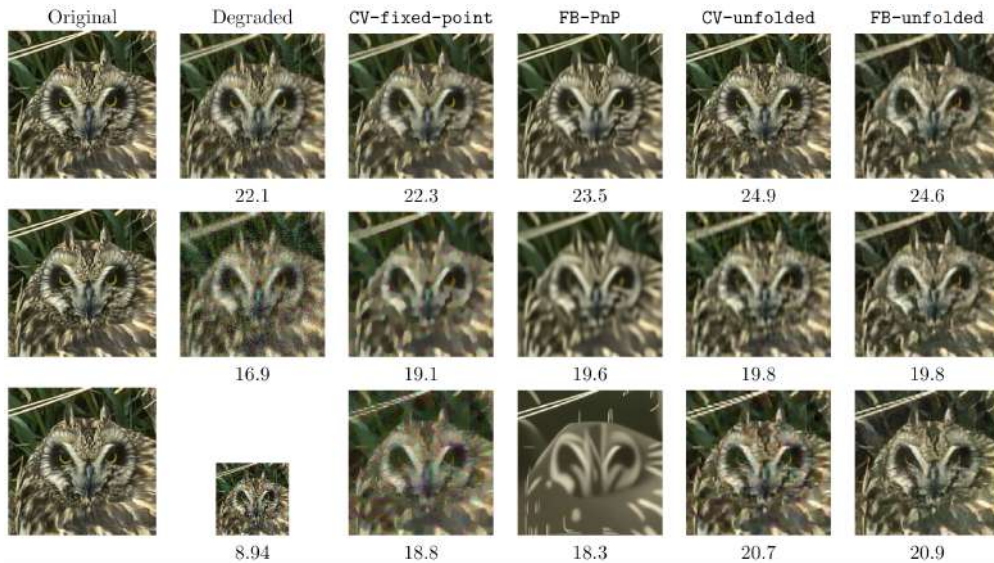
Forward model: Link between degradation \mathcal{D} and data fidelity h_z

FORWARD MODEL: $z = \mathcal{D}(\mathbf{A}\bar{x})$

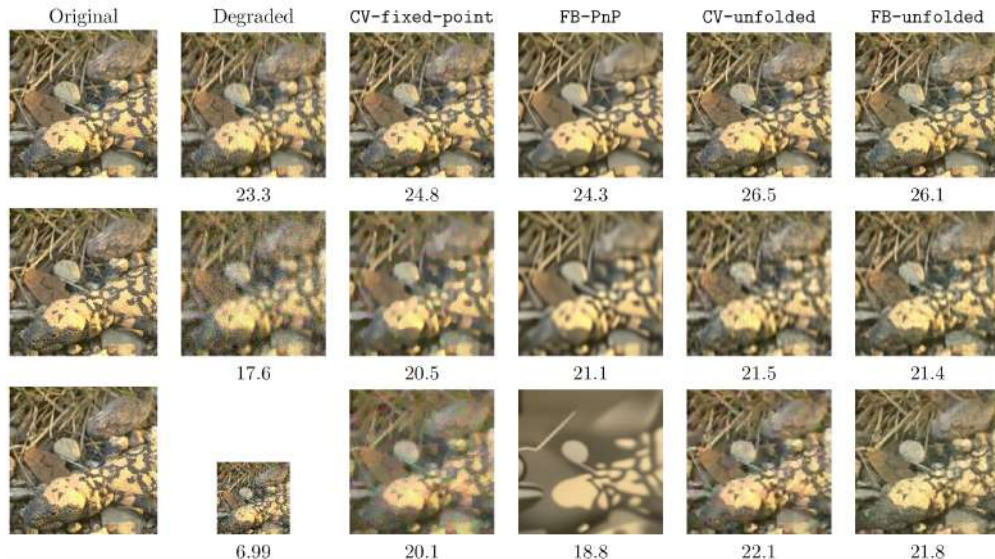
VARIATIONAL FORMULATION: Define the estimate \hat{x} as $\mathbf{0} \in \partial h_z(\hat{x}) + \lambda \partial g(\hat{x})$

- **Additive white Gaussian noise (AWGN)**
- **Coloured Gaussian noise**
- **Poisson noise**
- **Uniformly bounded noise**
 - \mathcal{D} introduces a bounded noise in the sense that there exists $\varepsilon > 0$ such that $\|\mathbf{A}\bar{x} - z\|^2 \leq \varepsilon$
 - $h_z(x) = \iota_{\mathcal{B}_2(z, \varepsilon)}(\mathbf{A}x)$ (*Morozov formulation*)
- **etc.**

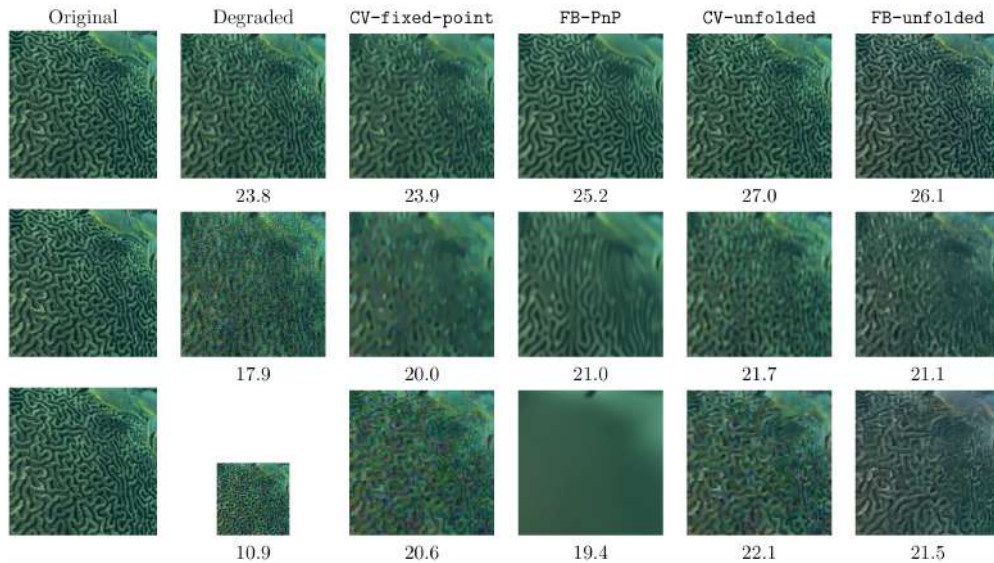
Problem solvers: Results



Problem solvers: Results



Problem solvers: Results



Python Toolbox: Based on DeepInverse and Pytorch

Goal: Find $\hat{\Theta} \in \underset{\Theta}{\text{Argmin}} \frac{1}{|\mathbb{I}|} \sum_{j \in \mathbb{I}} \ell(\bar{x}_j, \mathcal{L}_{\Theta}^K(z_j))$.

The screenshot shows the PyTorch website's 'Get Started' section. At the top, there's a navigation bar with links for 'Learn', 'Community', 'Projects', 'Docs', 'Blog & News', and 'About', along with a 'JOIN' button and a search icon. The main heading is 'Get Started', followed by the text: 'Select preferences and run the command to install PyTorch locally, or get started quickly with one of the supported cloud platforms.' Below this is a row of buttons: 'Start Locally', 'PyTorch 2.x', 'Start via Cloud Partners', 'Previous PyTorch Versions', and 'PyTorch for Edge'. The 'Start Locally' button is selected. Underneath, there's a 'Shortcuts' sidebar with links to 'Prerequisites', 'macOS Version', 'Python', 'Package Manager', 'Installation', 'pip', 'Verification', 'Building from source', and 'Prerequisites'. The main content area is titled 'Start Locally' and contains instructions: 'Select your preferences and run the install command. Stable represents the most currently tested and supported version of PyTorch. This should be suitable for many users. Preview is available if you want the latest, not fully tested and supported, builds that are generated nightly. Please ensure that you have met the prerequisites below (e.g., numpy), depending on your package manager. You can also install previous versions of PyTorch. Note that LibTorch is only available for C++.' A 'NOTE' states: 'Latest PyTorch requires Python 3.9 or later.' At the bottom, there's a table with columns for 'PyTorch Build', 'Your OS', and 'Windows'. The 'PyTorch Build' column has 'Stable (2.8.0)' and 'Preview (Nightly)'. The 'Your OS' column has 'Linux', 'Mac', and 'Windows'.

PyTorch

Learn Community Projects Docs Blog & News About JOIN

Get Started

Select preferences and run the command to install PyTorch locally, or get started quickly with one of the supported cloud platforms.

Start Locally PyTorch 2.x Start via Cloud Partners Previous PyTorch Versions PyTorch for Edge

Start Locally

Select your preferences and run the install command. Stable represents the most currently tested and supported version of PyTorch. This should be suitable for many users. Preview is available if you want the latest, not fully tested and supported, builds that are generated nightly. Please ensure that you have met the prerequisites below (e.g., numpy), depending on your package manager. You can also install previous versions of PyTorch. Note that LibTorch is only available for C++.

NOTE: Latest PyTorch requires Python 3.9 or later.

PyTorch Build	Your OS	Windows
Stable (2.8.0)	Linux	Mac
Preview (Nightly)		

Python Toolbox: Based on DeepInverse and Pytorch

Goal: Design \mathcal{L}_{Θ}^K



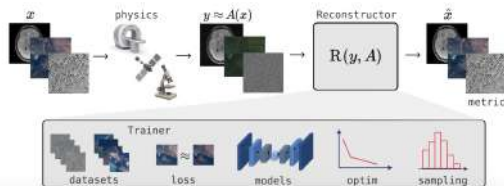
[Quickstart](#) [Examples](#) [User Guide](#) [API](#) [Finding Help](#) [More](#) *

DeepInverse: a Python library for imaging with deep learning

[Test](#) [Install](#) [Build docs](#) [Python 3.10+](#) [Code style](#) [Black](#) [Codecov](#) [82%](#) [pip install 1.0k/march](#)
[DeepInver library](#) 179 members [Open in Colab](#)

DeepInverse is an open-source PyTorch-based library for solving imaging inverse problems with deep learning. [deepinv](#) accelerates deep learning research across imaging domains, enhances research reproducibility via a common modular framework of problems and algorithms, and lowers the entrance bar to new practitioners.

GitHub: [deepinv/deepinv](#)



Python Toolbox: Based on DeepInverse and Pytorch

Goal: Design \mathcal{L}_{Θ}^K

The screenshot displays the DeepInverse website interface. At the top, the navigation bar includes links for Quickstart, Examples (which is highlighted), User Guide, API, Finding Help, and a More menu. A search bar is located on the right side of the navigation bar. Below the navigation bar, the main content area is titled '5 minute quickstart tutorial'. A blue note box at the top of the main content area states: 'New to DeepInverse? Get started with the basics with the 5 minute quickstart tutorial.' The main heading is '5 minute quickstart tutorial', followed by the text 'Follow this example to get started with DeepInverse in under 5 minutes.' Below this, a 'Contents' section lists five items: 1. [Install](#), 2. [Physics](#), 3. [Models](#), 4. [Datasets](#), and 5. [What's next](#). The first item, '1. Install', is expanded, showing the text 'First, install and import the latest stable release of `deepinv`:'.

Section Navigation

- Basics
 - 5 minute quickstart tutorial
 - Use a pretrained model
 - Use iterative reconstruction algorithms
 - Bring your own dataset
 - Bring your own physics
- Models
- Physics
- Optimization
- Plug-and-Play
- Diffusion & MCMC
- Unfolded
- Self-Supervised Learning
- Adversarial Learning
- External Libraries

Quickstart **Examples** User Guide API Finding Help More

Search

On this page

1. Install
2. Physics
3. Models
4. Datasets

What's next?

This Page

- Show Source

[Download source code](#)

[Download Jupyter notebook](#)

[Download zipped](#)

5 minute quickstart tutorial

Follow this example to get started with DeepInverse in under 5 minutes.

Contents

1. [Install](#)
2. [Physics](#)
3. [Models](#)
4. [Datasets](#)
5. [What's next](#)

1. Install

First, install and import the latest stable release of `deepinv`:

Python Toolbox

<https://perso.ens-lyon.fr/nelly.pustelnik/PNN/>

Model-based neural networks

[Proximal algorithms](#)[Plug-and-Play](#)[Unfolded](#)[Contacts](#)

This webpage provides basics codes to perform image reconstruction tasks with [DeepInverse library](#). To install it, follow the instructions provided [at this link](#).

We provide the codes necessary to reproduce part of the experiments detailed in our EUSIPCO tutorial "Proximal Neural Networks: Wedding Variational Methods and Artificial Intelligence" and additional codes to deepen the understanding. Utilizing the DeepInverse library enables us to concentrate on the structure of iterative schemes while leveraging established knowledge about data-term and prior design, as well as their associated gradient and proximity operators.

Proximal algorithms

- **Forward-backward**
 - Example in image restoration (Blur + Gaussian noise) with TV-L12 denoiser: [\[Code Python\]](#) [\[Notebook Google Colab\]](#)
- **FISTA**
 - Example in image restoration (Blur + Gaussian noise) with TV-L12 denoiser: [\[Code Python\]](#) [\[Notebook Google Colab\]](#)
- **Douglas-Rachford**
 - Example in image restoration (Blur + Gaussian noise) with TV-L12 denoiser: [\[Code Python\]](#)
- **Loris-Verhoeven**
 - Example in image restoration (Blur + Gaussian noise) with TV-L12 denoiser: [\[Code Python\]](#)
- **Condat-Vu**
 - Example in image restoration (Blur + Gaussian noise) with TV-L1 denoiser: [\[Code Python\]](#)
 - Example in image restoration (Blur + Gaussian noise) with TV-L12 denoiser: [\[Code Python\]](#) [\[Notebook Google Colab\]](#)
- **Chambolle-Pock**
 - Example in image restoration (Blur + Gaussian noise) with TV-L12 denoiser: [\[Code Python\]](#)

Plug-and-play