On learning optimized reaction diffusion processes for effective image restoration

Badr Soulaimani
Dhia Znaidi
Ilias Elfryakh
Rayen Ben Ismail
Wassim Chakroun

28/02/2023

Table of contents

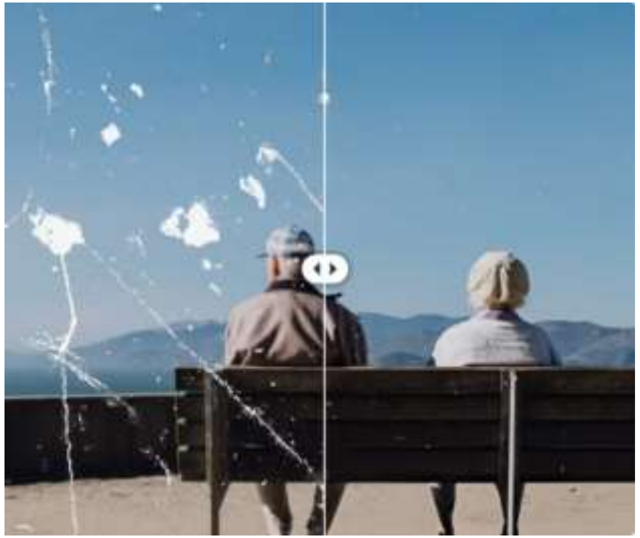
Problematic

Image restoration:

The process of recovering or improving the quality of a degraded or distorted image.

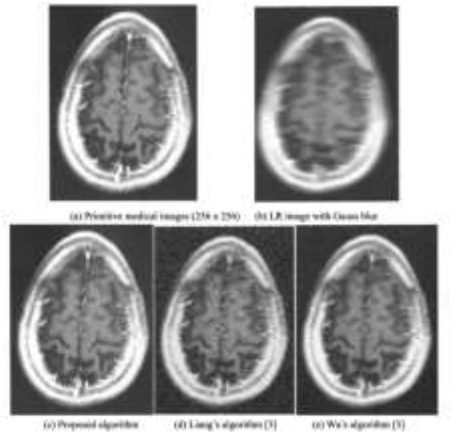
Problem:

Image restoration with high computational efficiency and restoration quality

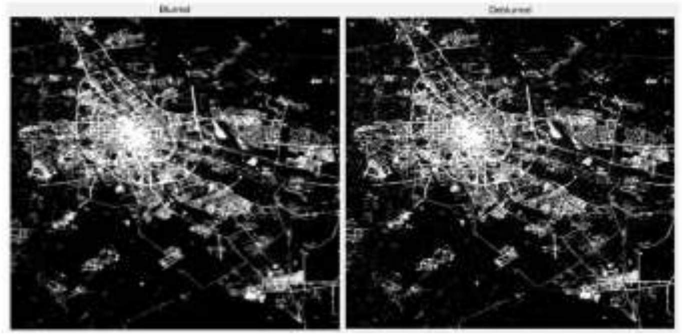


Problematic

Applications:


(a) Primitive medical images (256 x 256)
(b) LR image with Gauss blur
(c) Proposed algorithm
(d) Liang's algorithm [3]
(e) Wu's algorithm [5]

Medical imaging



Satellite imaging

Digital photography

Non-linear diffusion process

Goals:

- Enhance or restore images
- Selectively smooth or sharpen certain regions of an image
- Preserve important features such as edges and textures

The diffusion process is controlled by a function:

Amount of diffusion that occurs at each point

Conventional nonlinear reaction-diffusion models

Definition:

Reaction-diffusion models are used for tasks such as:

- image denoising
- image segmentation
- image enhancement

The models work by simulating the behavior of a set of interacting substances that diffuse through an image. The substances are typically modeled as concentration fields, and their behavior is governed by a set of nonlinear partial differential equations

Penalty, influence & edge-stopping functions

Penalty functions

- penalize the deviation from a desired behavior in the diffusion process
- enforce certain constraints on the diffusion process, such as limiting the diffusion in certain regions of the image

Influence functions

- control the diffusion coefficients in the diffusion process
- determine how much diffusion occurs in different parts of the image

Edge-stopping functions

- control the strength of the diffusion process at edges
- reduce the diffusion coefficient at edges to prevent over-smoothing

Lasso Regularization(L1)

$$loss = \sum_{i=0}^{n}(y_i - X_i\beta)^2 + \sum_{j=0}^{m}|\beta_j|$$

Ridge Regularization(L2)

$$loss = \sum_{i=0}^{n}(y_i - X_i\beta)^2 + \sum_{j=0}^{m}\beta_j^2$$

Squared Error Loss:

$$loss = \sum_{i=0}^{n}(y_i - \hat{y})^2$$

$$loss = \sum_{i=0}^{n}(y_i - X_i\beta)^2$$

$where: y_i = actual, \hat{y} = predicted,$
$X = input, \beta = coefficient$

Commonly used penalty functions

c(|∇u|) = 1 / (1 + (|∇u|/K)^2)

where K is a constant that controls the degree of diffusion.

P-M influence function

g(x, y) = 1 / (1 + (|∇f(x, y)|/T)^2)

where f(x,y) is the input image, ∇f(x,y) is the gradient of the image at pixel (x,y), and T is a threshold parameter.

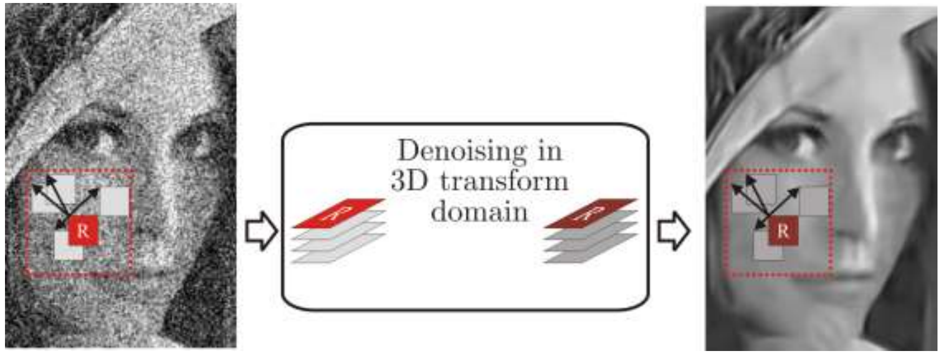
Canny edge-stopping function

State-of-the-art Methods

Block-matching and 3D filtering(BM3D):

Limitations:

computationally expensive / a large amount of memory and processing power.



State-of-the-art Methods

Cascade of Shrinkage Fields (CSF):

Limitations:

CSF assumes that the data term is quadratic and that the operator A can be interpreted as a convolution operation, which limits its applicability.

State-of-the-art Methods

Perona-Malik diffusion:

non linear diffusion models ⇒ very impressive results for image processing

Biased anisotropic diffusion (reaction diffusion) a variant introduces a bias term ⇒ free the user from the difficulty of specifying an appropriate stopping time for the P-M diffusion process.

$$\frac{u_{t+1} - u_t}{\Delta t} = -\sum_{i=\{x,y\}} \nabla_i^\top \Lambda(u_t) \nabla_i u_t \doteq -\sum_{i=\{x,y\}} \nabla_i^\top \phi(\nabla_i u_t)$$

$$\Lambda(u_t) = \operatorname{diag}\left(g\left(\sqrt{(\nabla_x u_t)_p^2 + (\nabla_y u_t)_p^2}\right)\right)_{p=1,\cdots,N}$$

Proposed model

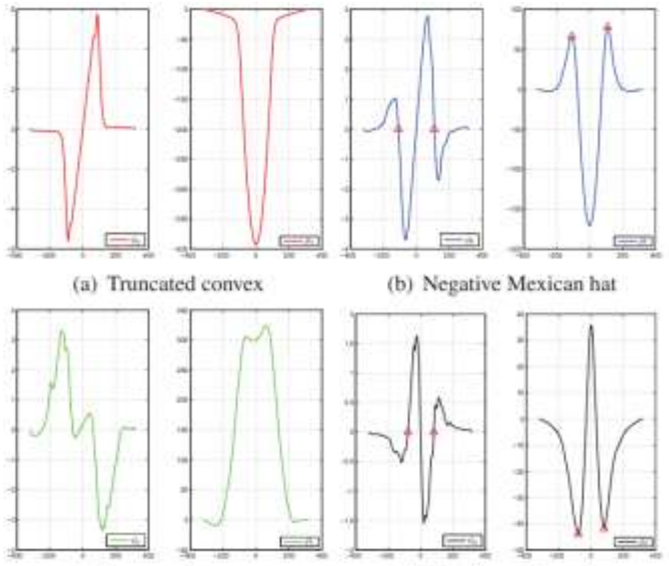
- The basic idea of our approach ⇒ learn PDEs from training data via an optimal control approach
- Proposed approach = conventional nonlinear reaction diffusion models + several parametrized linear filters + parametrized influence functions

$$\frac{u_t - u_{t-1}}{\Delta t} = \underbrace{-\sum_{i=1}^{N_k} K_i^{t\top} \phi_i^t(K_i^t u_{t-1})}_{\text{diffusion term}} - \underbrace{\psi(u_{t-1}, f_n)}_{\text{reaction term}}. \qquad K_i u \Leftrightarrow k_i * u.$$

Learned penalty functions

A major finding in this paper is that our learned penalty functions significantly differ from the usual penalty functions

- Learned functions have multiple minima different from zero and hence are able to enhance certain image structures.



(a) Truncated convex (b) Negative Mexican hat

(c) Truncated concave (d) Double-well penalty

RBFs and influence function

- Radial basis function (RBF) is a real-valued function whose value depends only on the distance between the input and some fixed point. They are typically used to build up function approximations.

- Here , the influence function is approximated via standard radial basis functions (RBFs) :

$$\phi_i^t(z) = \sum_{j=1}^{M} w_{ij}^t \varphi\left(\frac{|z-\mu_j|}{\gamma_j}\right)$$

$$\Downarrow$$

$$\phi_i^t(y) = G(y) \cdot w_i^t$$

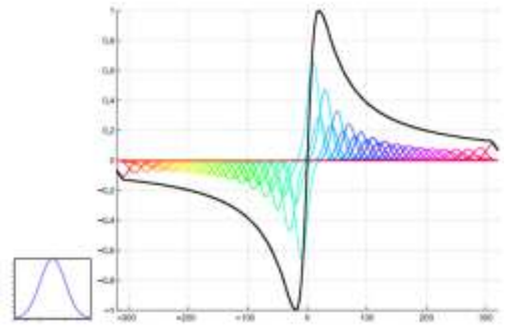
$$\Rightarrow \underbrace{\begin{bmatrix} \varphi(\frac{|y_1-\mu_1|}{\gamma}) & \varphi(\frac{|y_1-\mu_2|}{\gamma}) & \cdots & \varphi(\frac{|y_1-\mu_M|}{\gamma}) \\ \varphi(\frac{|y_2-\mu_1|}{\gamma}) & \varphi(\frac{|y_2-\mu_2|}{\gamma}) & \cdots & \varphi(\frac{|y_2-\mu_M|}{\gamma}) \\ \vdots & \vdots & \ddots & \vdots \\ \varphi(\frac{|y_Q-\mu_1|}{\gamma}) & \varphi(\frac{|y_Q-\mu_2|}{\gamma}) & \cdots & \varphi(\frac{|y_Q-\mu_M|}{\gamma}) \end{bmatrix}}_{G(y)} \underbrace{\begin{bmatrix} w_{i1} \\ w_{i2} \\ \vdots \\ w_{iM} \end{bmatrix}}_{w_i} = \underbrace{\begin{bmatrix} \phi_i(y_1) \\ \phi_i(y_2) \\ \vdots \\ \phi_i(y_Q) \end{bmatrix}}_{\phi_i(y)}$$

RBFs and penalty function

- In this paper, we investigate two typical RBFs :
  - Gaussian radial basis φ_g
  - Triangular-shaped radial basis φ_t

$$\varphi_g(z) = \varphi\left(\frac{|z-\mu|}{\gamma}\right) = \exp\left(-\frac{(z-\mu)^2}{2\gamma^2}\right)$$

$$\varphi_t(z) = \varphi\left(\frac{|z-\mu|}{\gamma}\right) = \begin{cases} 1 - \frac{|z-\mu|}{\gamma} & |z-\mu| \leq \gamma \\ 0 & |z-\mu| > \gamma \end{cases}$$



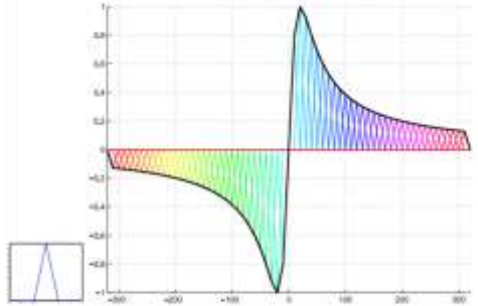
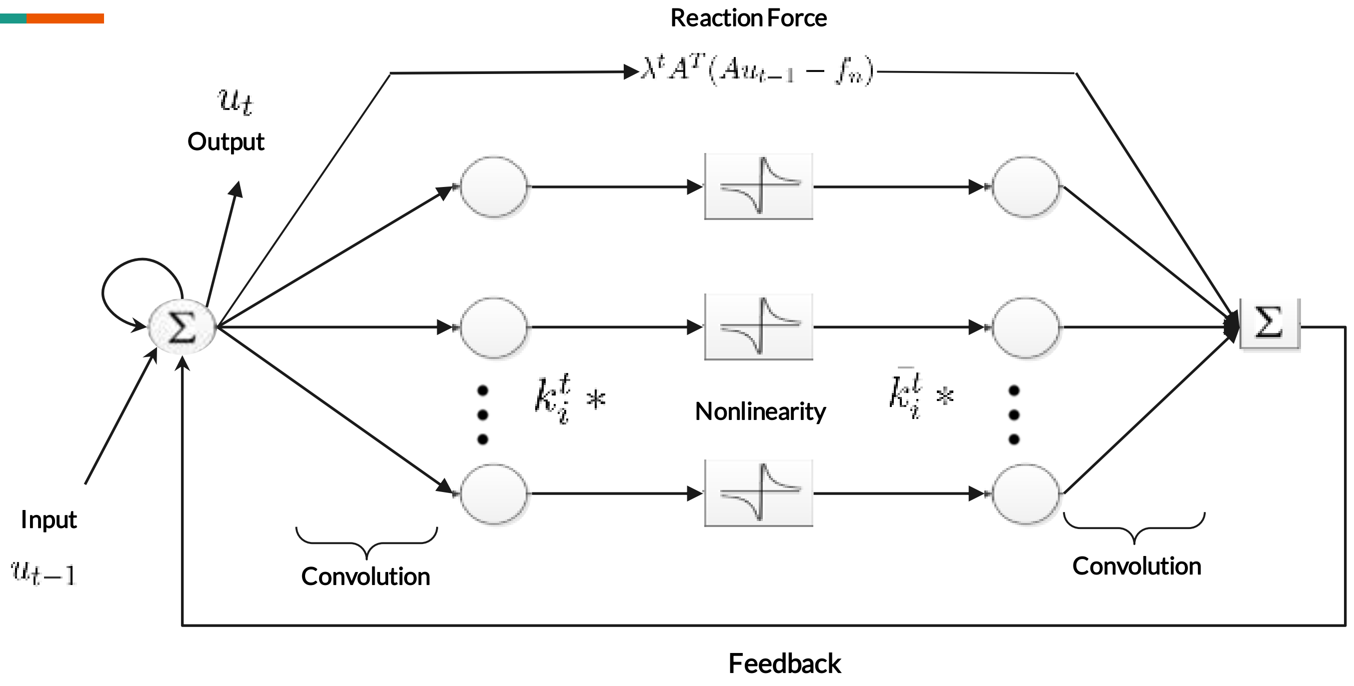


Illustration of the training process



$$u_t = u_{t-1} - \left(\sum_{i=1}^{N_k} \bar{k}_i^t * \phi_i^t(k_i^t * u_{t-1}) + \lambda^t A^T(Au_{t-1} - f_n)\right)$$

Image Boundary Condition : Intuition

- Boundary conditions refer to the way in which the values at the edges of an image are treated during the diffusion process.
- In the case of the our model, the authors apply a symmetric boundary condition, which means that the values at the edges of the image are mirrored across the boundary.

$$\begin{matrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{matrix} \quad \Longrightarrow \quad \begin{matrix} 5 & 4 & 5 & 6 & 5 \\ 2 & 1 & 2 & 3 & 2 \\ 5 & 4 & 5 & 6 & 5 \\ 8 & 7 & 8 & 9 & 8 \\ 5 & 4 & 5 & 6 & 5 \end{matrix}$$

Image Boundary Condition

$$u_t = u_{t-1} - \left(\sum_{i=1}^{N_k} K_i^{t^\top} \phi_i^t (K_i^t u_{t-1}) + \lambda^t (u_{t-1} - f_n) \right) \Longleftrightarrow u_t = u_{t-1} - (\sum_{i=1}^{N_k} \bar{k}_i^t * \phi_i^t (k_i^t * u_{t-1}) + \lambda^t (u_{t-1} - f_n))$$

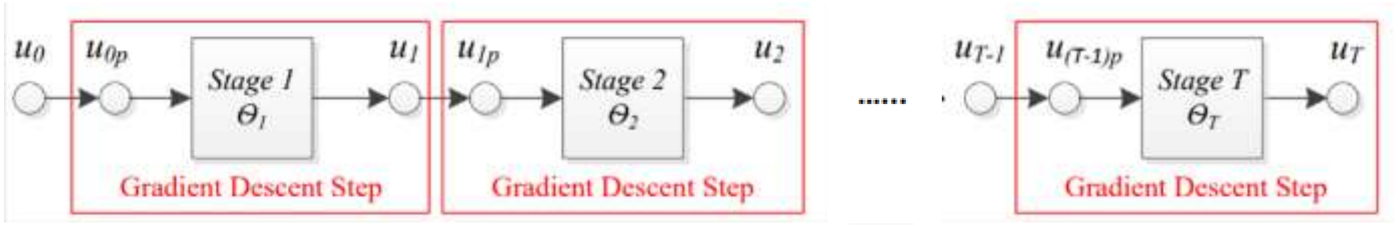
$$K_i u \Leftrightarrow k_i * u.$$

- Some imperfections at the image boundary are introduced
- $K^\top v$ can be interpreted as the convolution with the kernel $\bar{k}$ only in the central region of image v. => This interpretation does not hold for the image boundary

- In the 2nd formulation, the convolution kernel $\bar{k}$ is applied to the whole image, thus bringing some artifacts at the boundary.
- the revised model is more tractable in practice, especially for training, as everything can be done by the convolution operation efficiently.

Image Boundary Condition

- Removing boundary artifacts :
  - The input u_{t-1} for stage t and f_n are padded using a sparse "padding" P matrix
  - After a diffusion step, we only crop the central region of the output image for usage using the sparse "cropping" matrix T
- Denoting $P_T = P \times T$ $u_{tp} = P_T u_t$,the exact diffusion process is illustrated below :



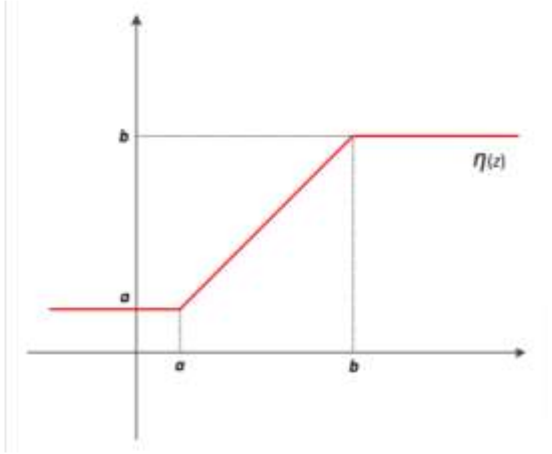
$$u_t^s = u_{(t-1)p}^s - \left(\sum_{i=1}^{N_k} \bar{k}_i^t * \phi_i^t(k_i^t * u_{(t-1)p}^s) + \lambda^t (u_{(t-1)p}^s - f_{np}^s) \right)$$

Diffusion Process: Image Denoising

$$u_t^s = u^s_{(t-1)p} - \left(\sum_{i=1}^{N_k} \bar{k}_i^t * \phi_i^t(k_i^t * u^s_{(t-1)p}) + \lambda^t (u^s_{(t-1)p} - f^s_{np}) \right)$$

Diffusion Process: JPEG Deblocking

$$u_t = D^\top \mathrm{proj}_Q \left(D \left(u_{t-1} - \sum_{i=1}^{N_k} \bar{k}_i^t * \phi_i^t (k_i^t * u_{t-1}) \right) \right)$$



The projection function

$$\longrightarrow$$

$$\begin{cases} u_t = D^\top v \\ v = \eta(z) \\ z = Dy \\ y = u_{t-1} - \sum\limits_{i=1}^{N_k} \bar{k}_i^t * \phi_i^t (k_i^t * u_{t-1}) \end{cases}$$

Greedy and joint training

Greedy training

- Train an RDP and use its output as the input of the next RDP in the pipeline
- Optimization of every RDP independently of each other(stage by stage).

$$\Theta_t = \{\lambda^t, \phi_i^t, k_i^t\}$$

$$\begin{tikzcd}[column sep=tiny] & \mathcal{L}(\Theta_t) = \sum_{s=1}^{S} \ell(u_t^{(s)}, u_{gt}^{(s)}) \, , \arrow[dl] \arrow[dr] & \\ \ell(u_t^{(s)}, u_{gt}^{(s)}) = \frac{1}{2}\|u_t^{(s)} - u_{gt}^{(s)}\|_2^2 & & \ell(\hat{\mathbf{x}}; \mathbf{x}_{\text{gt}}) = -20\log_{10}\left(\frac{R\sqrt{D}}{\|\hat{\mathbf{x}} - \mathbf{x}_{\text{gt}}\|}\right) \end{tikzcd}$$

$$\frac{\partial \ell(u_t, u_{gt})}{\partial \Theta_t} = \frac{\partial u_t}{\partial \Theta_t} \cdot \frac{\partial \ell(u_t, u_{gt})}{\partial u_t}$$

Joint training

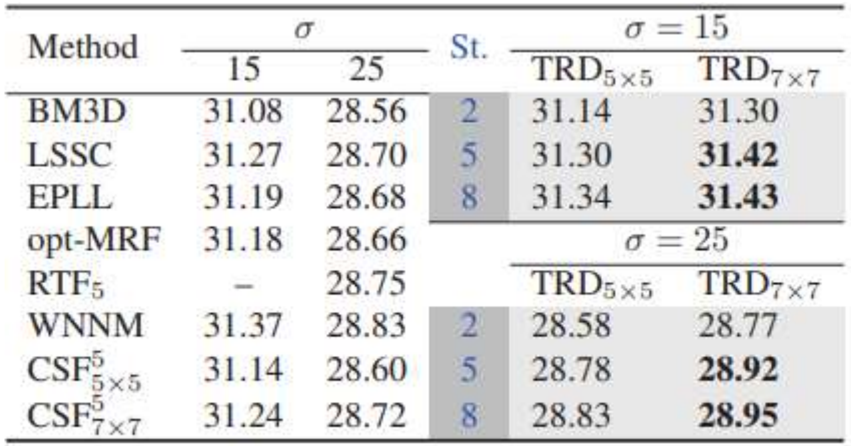
- Train RDPs and use its output as the input of the next RDP in the pipeline
- Optimization of all RDPs of pipeline simultaneously.

$$\mathcal{L}(\Theta_{1,\cdots,T}) = \sum_{s=1}^{S} \ell(u_T^{(s)}, u_{gt}^{(s)})$$

- Loss function depends only on the output of final stage.
- Optimization: Backtracking

Image denoising

- 400 training samples of size 180 * 180
- 20.8h training time for TRD5 (7*7) on a Matlab Implementation (200 L-BFGS iterations, 63 RBFs)
- A greedy training is done at first for up to 8 stage and then a joint training for a certain stage (5 or 7 here)
- Better results on both models compared to other methods.
- Increasing the size to 9 * 9 doesn't yield a big gain in performance (28.96 dB at 5 stages)
- Joint training always improves results by at least 0.1 dB in case of T >= 5
- Not very sensitive to initialization (deviation by about 0.01dB) especially for greedy training



Method	σ		St.	$\sigma = 15$	
	15	25		$\text{TRD}_{5\times5}$	$\text{TRD}_{7\times7}$
BM3D	31.08	28.56	2	31.14	31.30
LSSC	31.27	28.70	5	31.30	**31.42**
EPLL	31.19	28.68	8	31.34	**31.43**
opt-MRF	31.18	28.66		$\sigma = 25$	
RTF_5	–	28.75		$\text{TRD}_{5\times5}$	$\text{TRD}_{7\times7}$
WNNM	31.37	28.83	2	28.58	28.77
$\text{CSF}^5_{5\times5}$	31.14	28.60	5	28.78	**28.92**
$\text{CSF}^5_{7\times7}$	31.24	28.72	8	28.83	**28.95**

Table 1. Average PSNR (dB) on 68 images from [36] for image denoising with $\sigma = 15, 25$.

Image denoising

Training a 5*5 model with T=10 stages :

With fixed influence function for all stages : 28.47dB

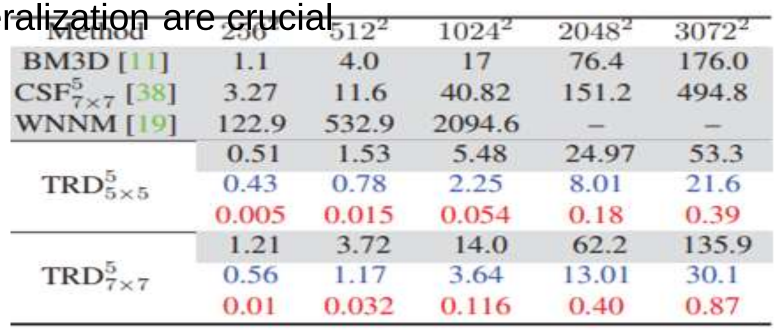
With optimized influence functions for all stages : 28.60dB

With fixed influence function for each stage : 28.56dB

With optimized influence functions for each stage : 28.86dB

-> Influence functions and generalization are crucial

TRD 5 (7×7) model outperforms all the recent state-of-the-arts on the exploited test dataset, while being the fastest method even with the CPU implementation.

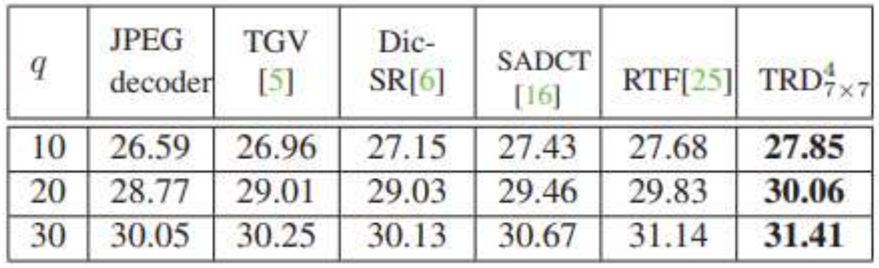


Method	256^2	512^2	1024^2	2048^2	3072^2
BM3D [11]	1.1	4.0	17	76.4	176.0
$CSF^5_{7\times7}$ [38]	3.27	11.6	40.82	151.2	494.8
WNNM [19]	122.9	532.9	2094.6	–	–
	0.51	1.53	5.48	24.97	53.3
$TRD^5_{5\times5}$	0.43	0.78	2.25	8.01	21.6
	0.005	0.015	0.054	0.18	0.39
	1.21	3.72	14.0	62.2	135.9
$TRD^5_{7\times7}$	0.56	1.17	3.64	13.01	30.1
	0.01	0.032	0.116	0.40	0.87

Table 2. Run time comparison for image denoising (in seconds)

JPEG deblocking

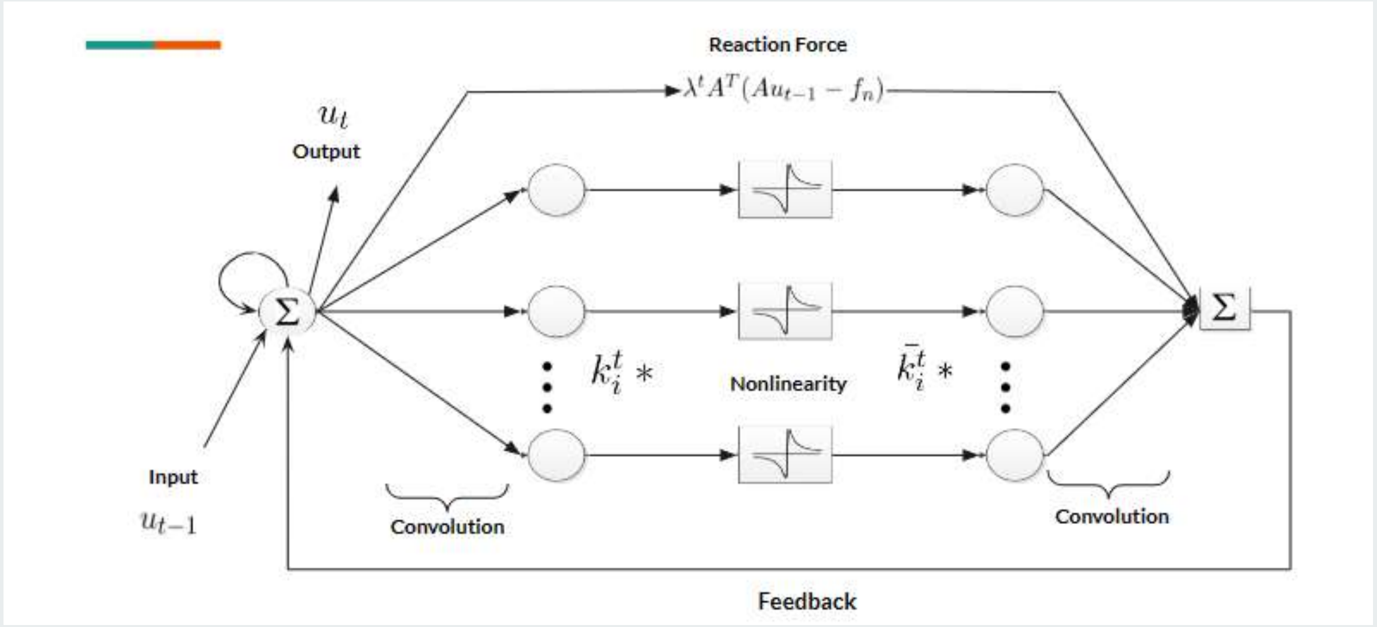
- Trained on 240 * 160 images distorted by JPEG blocking artifacts on three compression quality settings for the JPEG encoder
- 4 stages are enough for JPEG deblocking
- TRD 4 (7*7) outperforms all competing approaches in terms of PSNR
- It is also very fast. It takes 0.095s on GPU for a common 1024 * 1024 image, compared to the strongest competitor SADCT that takes about 56.5s on CPU.



q	JPEG decoder	TGV [5]	Dic-SR[6]	SADCT [16]	RTF[25]	$\text{TRD}^4_{7\times7}$
10	26.59	26.96	27.15	27.43	27.68	**27.85**
20	28.77	29.01	29.03	29.46	29.83	**30.06**
30	30.05	30.25	30.13	30.67	31.14	**31.41**

Table 3. JPEG deblocking results for natural images, reported with average PSNR values.

Wrap-up



Reaction Force
$\lambda^t A^T(Au_{t-1} - f_n)$
u_t
Output
$k_i^t *$
Nonlinearity
$\bar{k}_i^t *$
Σ
Σ
Input
u_{t-1}
Convolution
Convolution
Feedback

Implementation details

Our implementation

The author's implementation

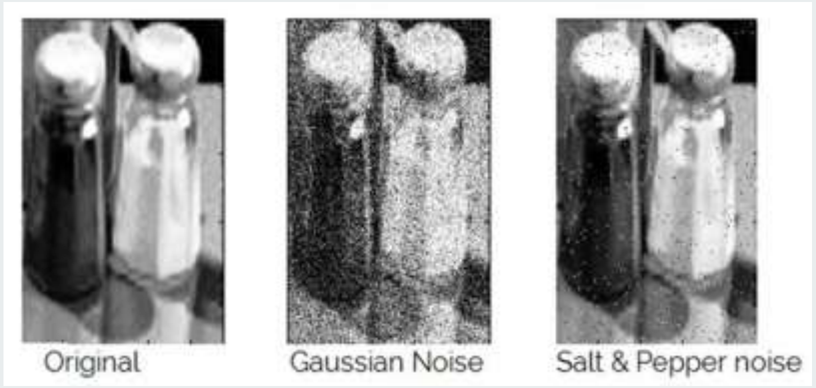
learning_optimized_reaction_diffusion.ipynb - Colaboratory (google.com)

jplumail/learning-image-restoration: Pytorch implementation of "On learning optimized reaction diffusion processes for effective image restoration" (github.com)

Used types of noise

Different types of noise:

- Gaussian noise
- Salt and pepper noise
- Uniformly distributed noise



Metrics

PSNR (Peak Signal to Noise Ratio):

measure of distortion used in digital images and allows to evaluate the reconstruction quality of the reconstructed image compared to the original image

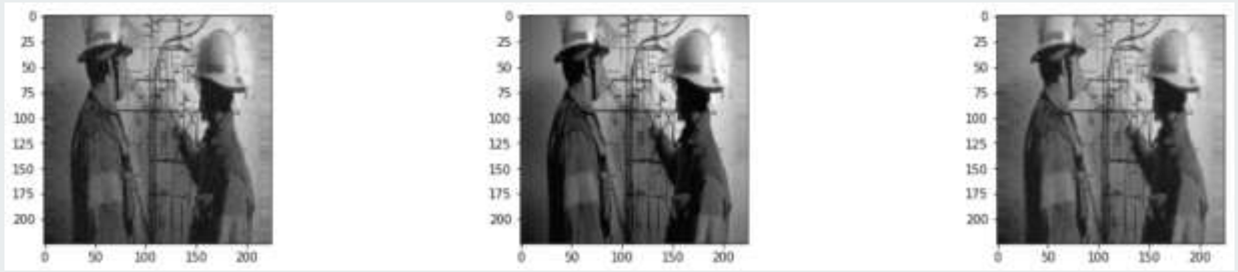
$$PSNR = 10 \cdot \log_{10}\left(\frac{d^2}{EQM}\right)$$

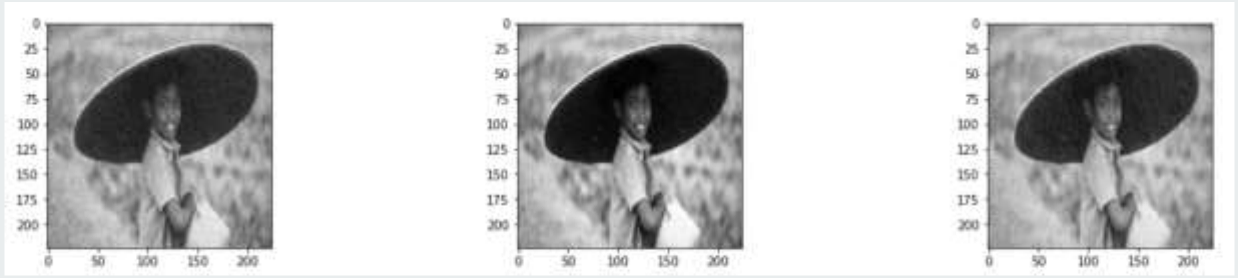
- d is the dynamic range of the signal (the maximum possible value for a pixel)

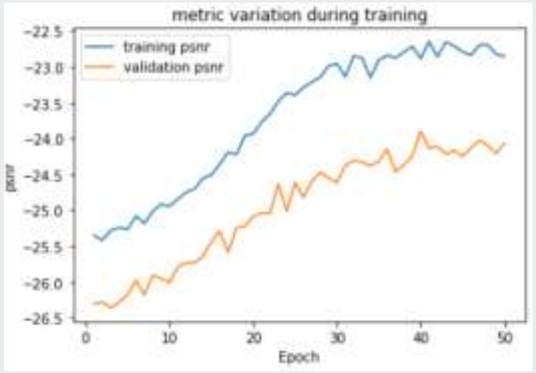
- MSE is the mean squared error

Gaussian noise

```
noise:
        mean = 0
        sigma = 0.05
epochs 100
```

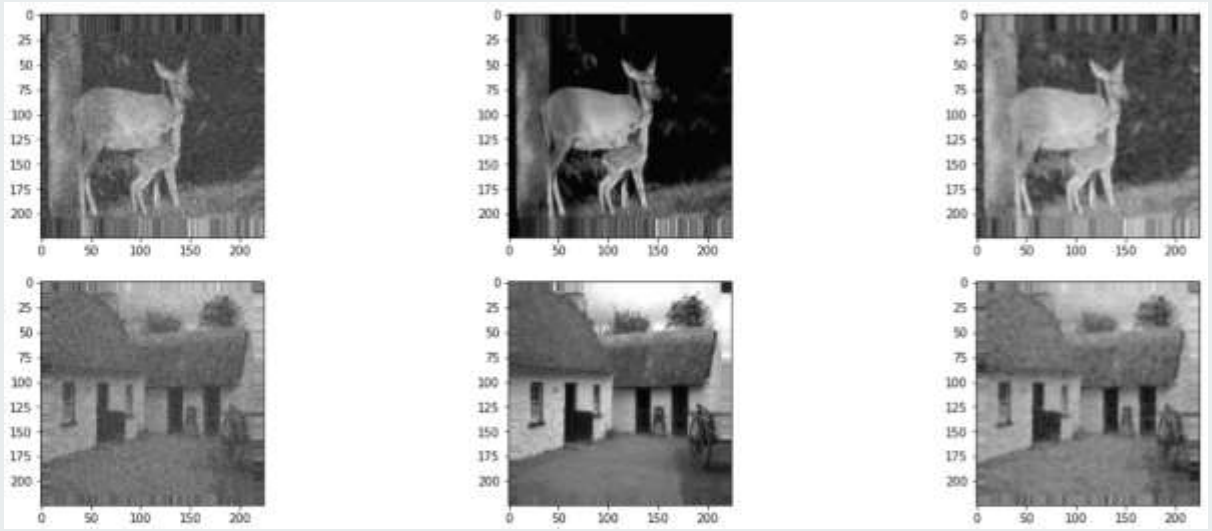


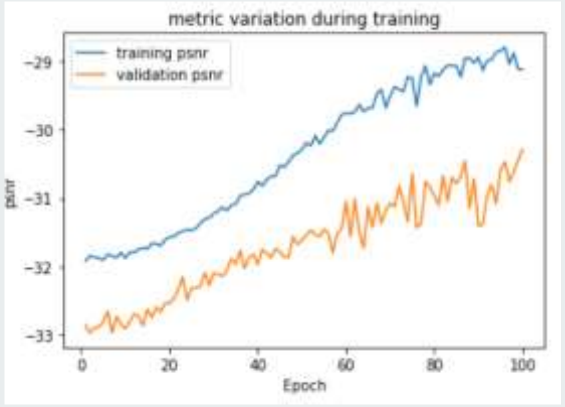




Gaussian noise

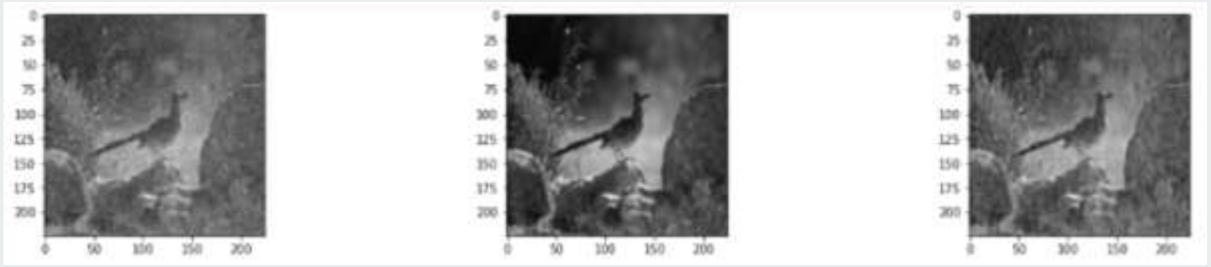
```
noise:
        mean = 0
        sigma = 0.15
epochs 100
```

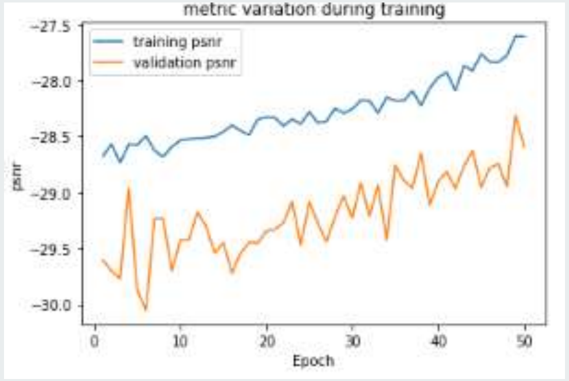




Gaussian noise

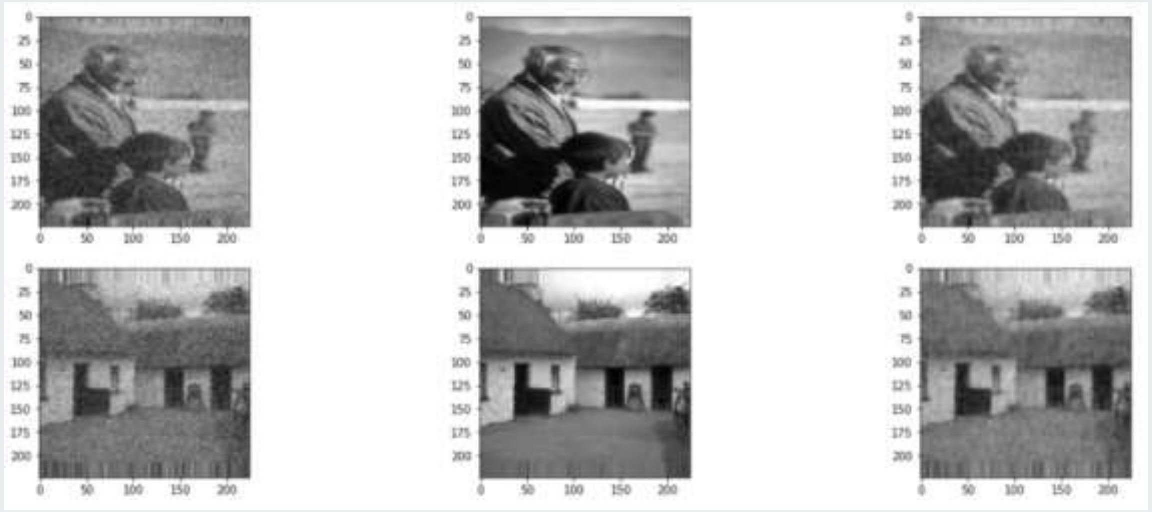
```
noise:
        mean = 0
        sigma =uniformly distributed
epochs 100
```

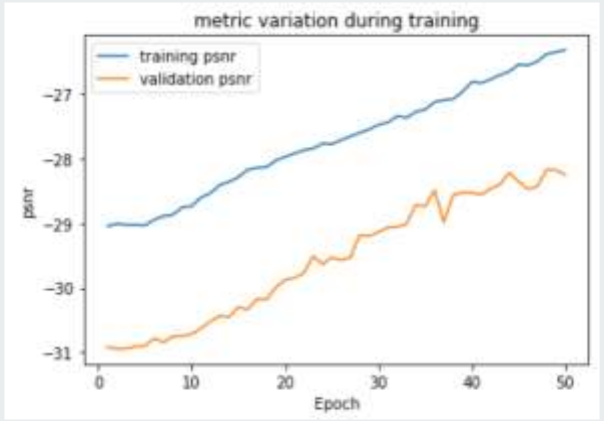




Uniformly distributed

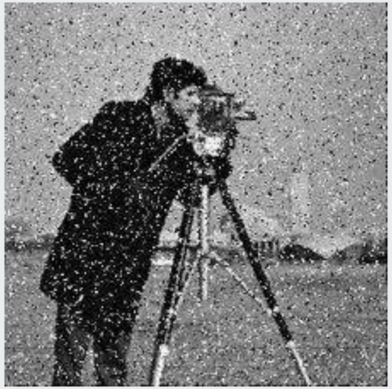
`noise: uniformly distributed`





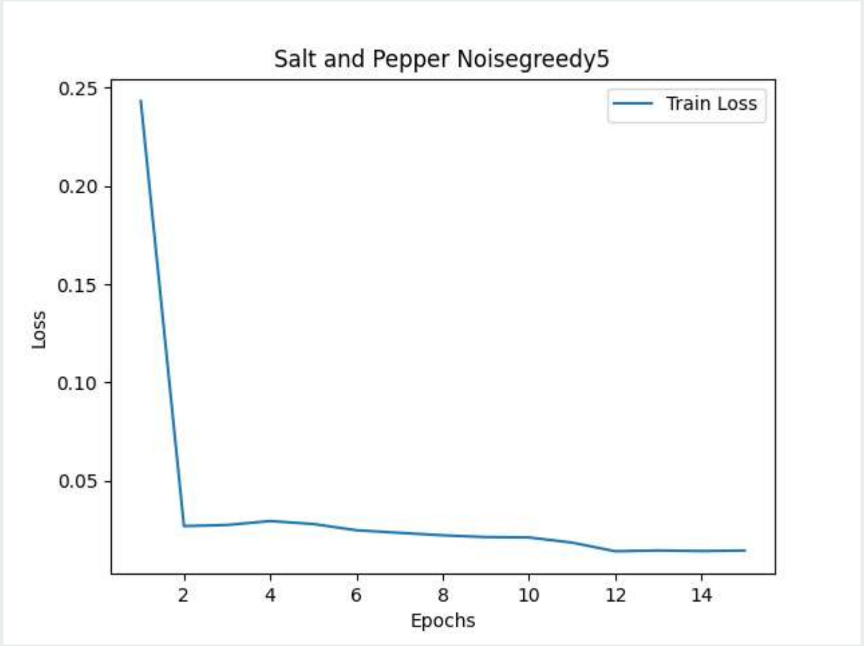
Salt & Pepper noise

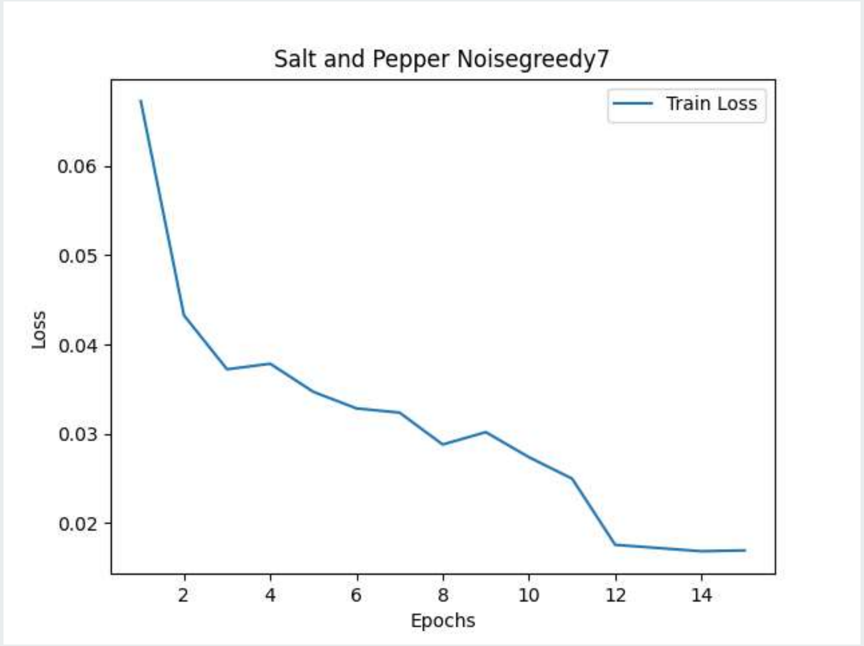
Salt-and-pepper noise, also known as impulse noise, is a form of noise sometimes seen on digital images. This noise can be caused by sharp and sudden disturbances in the image signal. It presents itself as sparsely occurring white and black pixels.



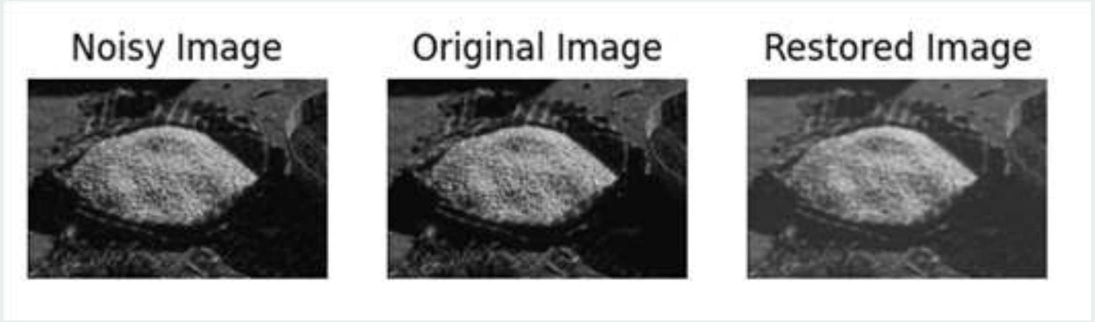
Salt & Pepper : Training

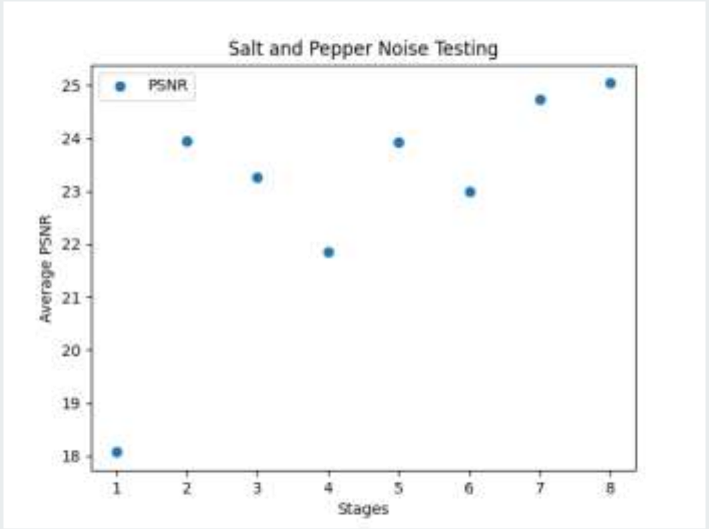
15 epochs/8 stages



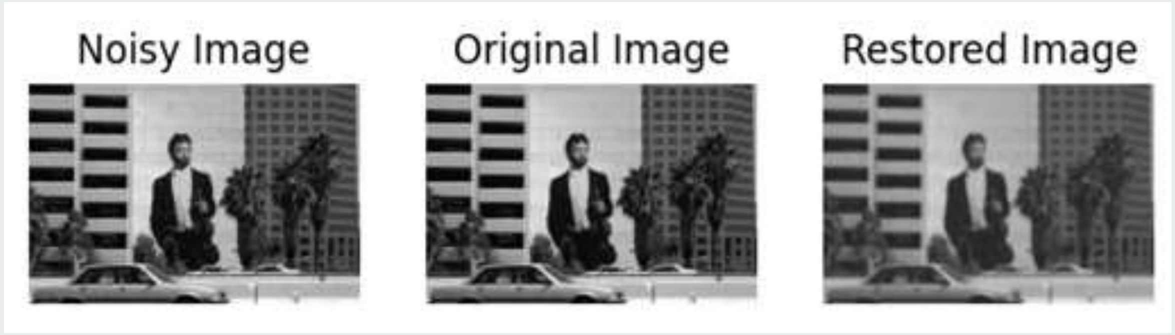


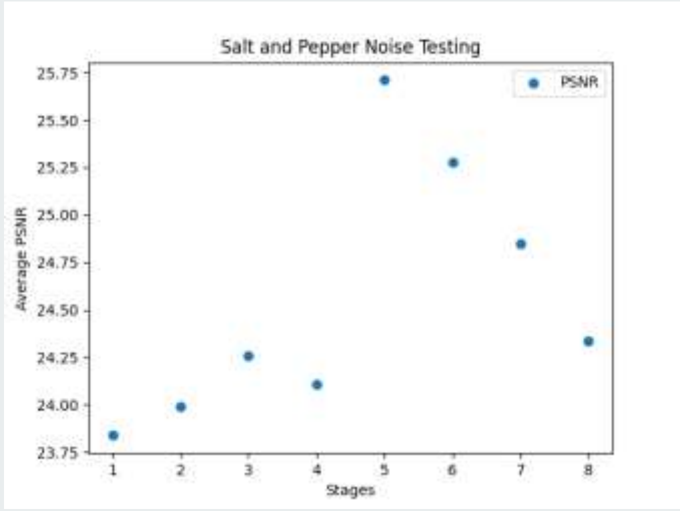
Salt & Pepper noise : Greedy 7





Salt & Pepper noise : Greedy 5

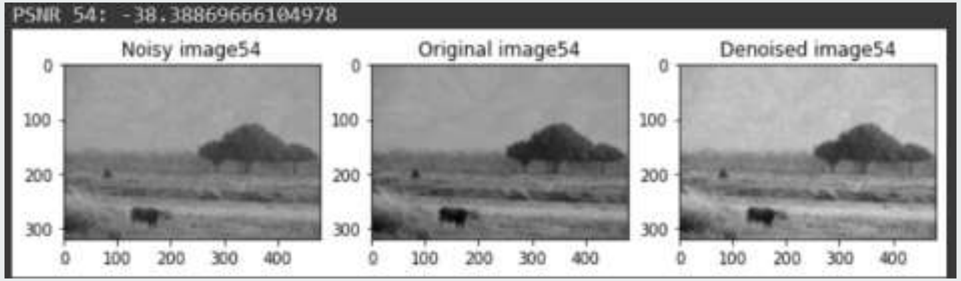




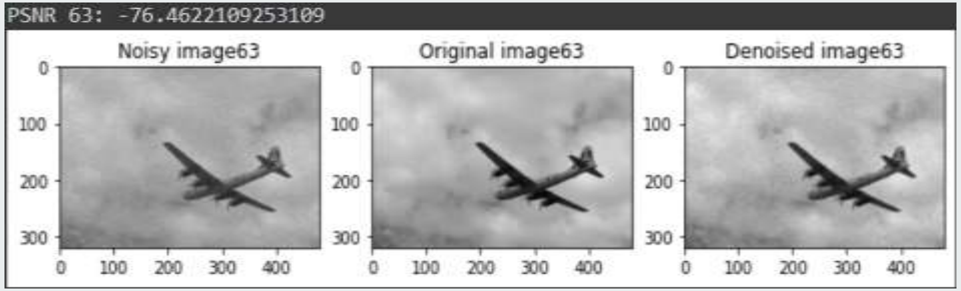
Reaction diffusion VS Heat equation for denoising

Performance on « BSDS300-images » with Heat equation for image denoising :

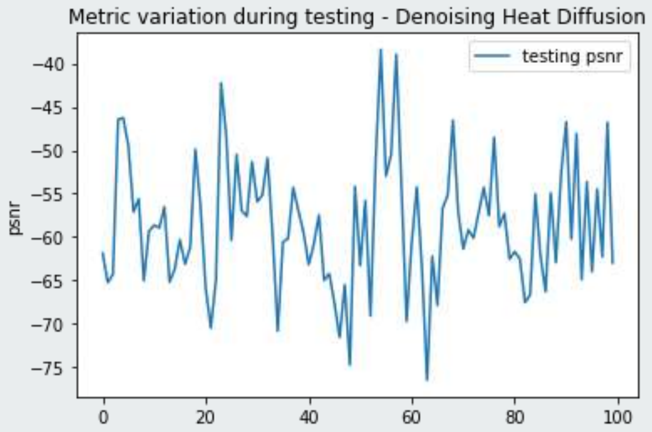
$$u^{(k+1)} = u^{(k)} - \lambda \left(2 \cdot \left(u^{(k)} - v\right) - \alpha \Delta u^{(k)}\right)$$



```
PSNR 54: -38.38869666104978
```



```
PSNR 63: -76.4622109253109
```



Conclusion