3A FISE - Spécialité Informatique - Majeures CIA & ISIA

Deep Avancé : Generative Models

Part I: patch-based methods

julien.Rabin (at) ensicaen.fr

Version : 3 Novembre 2025

Part I — Outline

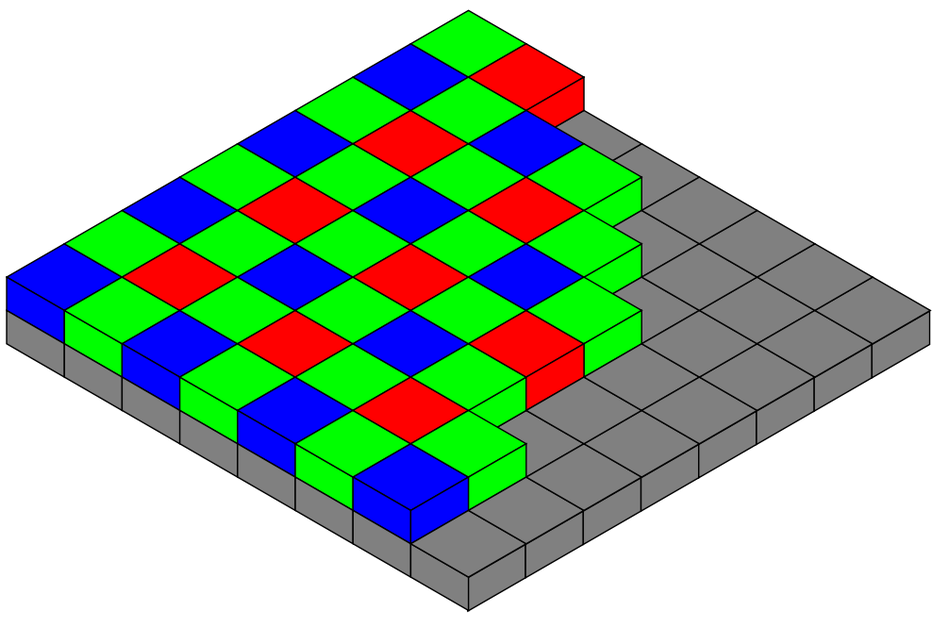
- **Image Filtering** with neighborhood methods
- filtering in deep neural network
- Application to **Image Editing**
- **Texture Synthesis**
- **Style transfer**

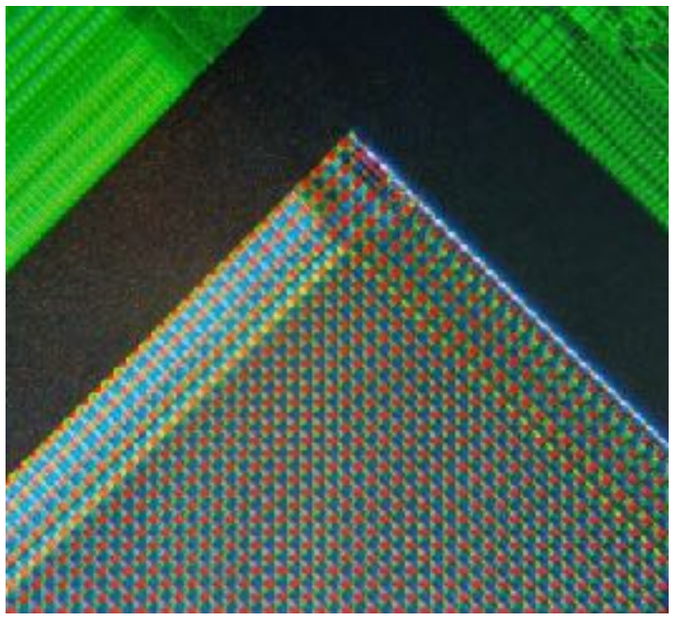
Image Filtering

Why ?

Image filtering

- has many direct applications, such as denoising and computational photography (enhancement, editing)
- is used in various signal processing tasks (reconstruction, detection, recognition, generation, etc).





How ?

In this course, we focus on **Neighborhood** based representation: spatial coordinates AND signal values (gray scales, colors, etc).

Examples of image filtering techniques:

- **Average** Filtering: Gaussian, box, (cross) Bilateral Filter, ...

- Other statistics: Median, morphological filters, Guided filtering (local regression), ...

- **Patch representation:** Non-Local Mean, PCA thresholding, BM3D ...

Notations

- Image $\mathfrak{u}$
  Spatial domain on a grid
  For color image $d{=}3$

$$u : x \in \Omega \to u(x) \in \mathbb{R}^d$$
$$\Omega \subset \mathbb{Z}^2$$
$$u(x) = (u_i(x))_{i=1,\dots,d}$$

- **Patch**
  Spatial size $w \times w$
  Dimension $D = d\,w^2$

$$p(x) = (u_i(x+\omega))_{i=1,\dots,d} \in \mathbb{R}^D$$
$$\omega = \{0,\dots,w-1\}^2$$

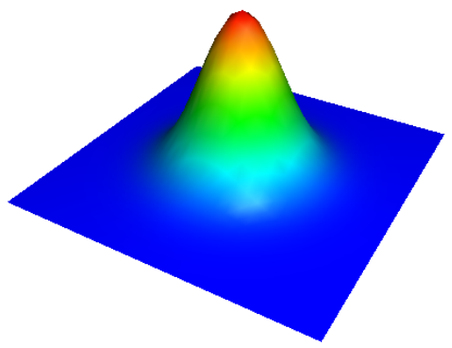
- Patch distance: $\|p(x) - p(y)\|^2 = \sum_{i=1}^{d} \sum_{t\in\omega} (u_i(x+t) - u_i(y+t))^2$

$$\|p(x) - p(y)\|^2 = \sum_{i=1}^{d} \sum_{t\in\omega} (p_i(x)|_t - p_i(y)|_t)^2$$

Gaussian Filter

- Related to *Heat Diffusion (see « traitement image avancé »)* $\frac{\partial u}{\partial t} = \rho \Delta u$
- Observed when image is *out of focus (depth estimation)*
- Application to linear scale space representation (e.g. SIFT 2000)
- Discrete Formulation: $\hat{u} = g_\sigma * u$

$$g_\sigma(x) = \frac{1}{C(x)} e^{-\frac{1}{2\sigma^2}\|x\|^2}$$



- $$\hat{u}(x) = \frac{1}{C(x)} \sum_{y \in \Omega} u(y) e^{-\frac{1}{2\sigma^2}\|x-y\|^2}$$

Technical details

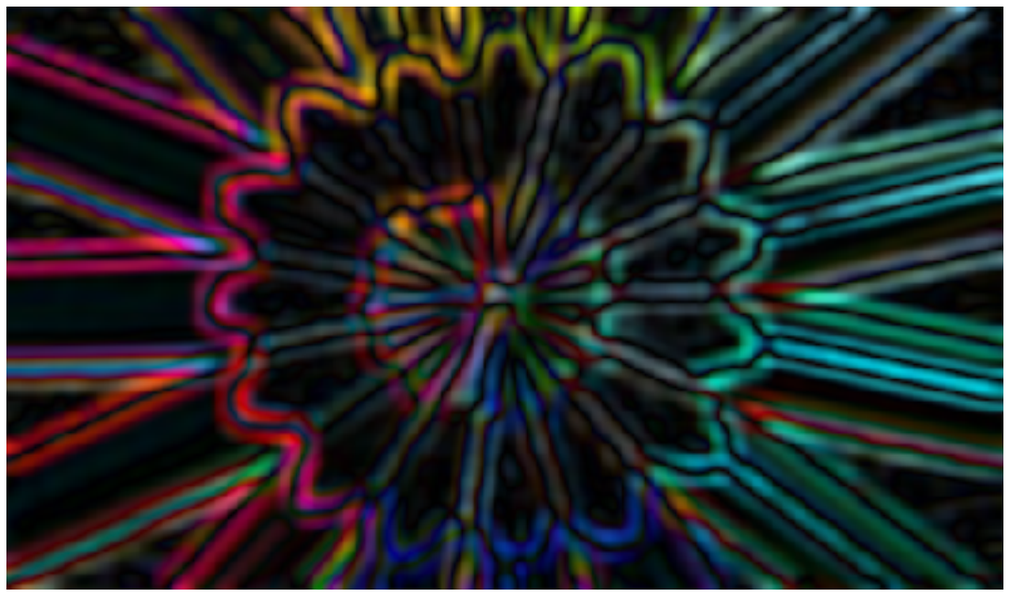
- Isotropy and truncation
- boundary conditions (Dirichlet, Neumann, Periodic, ...)
- What is C ? does it depend on x ?
- Color channels processing
- Time Complexity with naive implementation
- 1D separability, Fourier transform

$$\hat{u}(x) = \frac{1}{C} \sum_{y \in \Omega} u(y) e^{-\frac{1}{2\sigma^2} \|x-y\|^2}$$

Illustration

Remove high frequencies (contrast loss, smooth edges, texture removal, etc)

Gaussian smoothing



Difference of Gaussian (DoG) is an edge detector (analogy with human visual system)

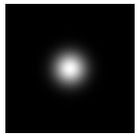
Box Filter

- Crude approximation of gaussian filtering... (*Sinc in Fourier domain !*)
- ... but very fast implementation (*Integral Image*)
- Popular in Neural Networks (*average pooling*)
- Application to scale space approximation (e.g. SURF in ECCV'06)
- Discrete Formulation: $\hat{u} = b_r * u$

$$b_r(x) = \frac{1}{(2r+1)^2} \mathbb{1}_{\|x\|_\infty < r} = \begin{cases} 1 & \text{if } \|x\|_\infty = \max(x_1, x_2) < r \\ 0 & \text{otherwise} \end{cases}$$

$$\hat{u}(x) = \frac{1}{(2r+1)^2} \sum_{\|y-x\|_\infty \leq r} u(y)$$

Illustration



Gaussian smoothing

Box filter (larger)

Box filter

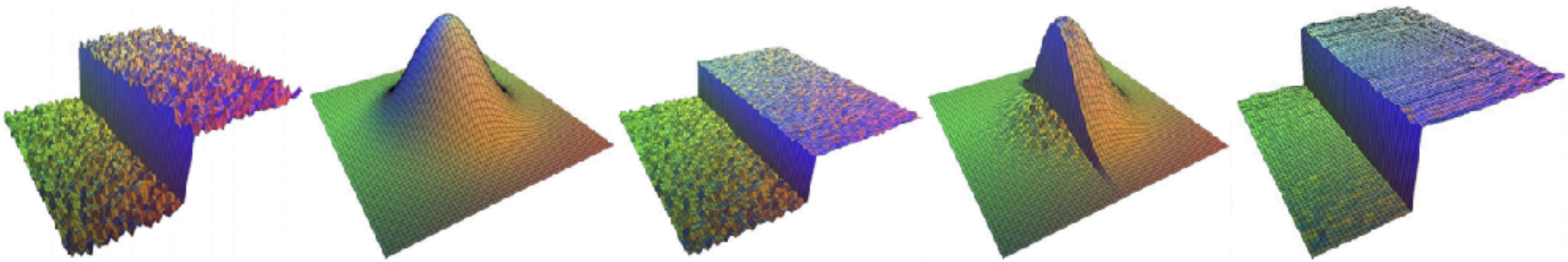
Bilateral Filter

- Gaussian kernel using **both space** and **color** coordinates (ICCV 1998)
- Preserve edges, remove texture
- Popular for image editing (Photoshop), more control (2 parameters)
- Discrete Formulation: not a convolution anymore ! (non linear)
- faster implementation / approximation exists

$$\hat{u}(x) = \frac{1}{C(x)} \sum_{y \in \Omega} u(y) \underbrace{e^{-\frac{1}{2\sigma^2}\|x-y\|^2}}_{\textit{prior (location)}} \underbrace{e^{-\frac{1}{2h^2}\|u(x)-u(y)\|^2}}_{\textit{classifier (color)}}$$

$$C(x) = \sum_{y \in \Omega} e^{-\frac{1}{2\sigma^2}\|x-y\|^2} e^{-\frac{1}{2h^2}\|u(x)-u(y)\|^2}$$

Bilateral Filter



input | spatial kernel f domain for the central pixel | influence g in the intensity for the central pixel | weight $f \times g$ | output

From *Fast Bilateral Filtering for the Display of High-Dynamic-Range Images*, Durand and Dorsey.

$$\hat{u}(x) = \frac{1}{C(x)} \sum_{y \in \Omega} u(y) \underbrace{e^{-\frac{1}{2\sigma^2}\|x-y\|^2}}_{\textbf{prior (location)}} \underbrace{e^{-\frac{1}{2h^2}\|u(x)-u(y)\|^2}}_{\textbf{classifier (color)}}$$

$$C(x) = \sum_{y \in \Omega} e^{-\frac{1}{2\sigma^2}\|x-y\|^2} e^{-\frac{1}{2h^2}\|u(x)-u(y)\|^2}$$

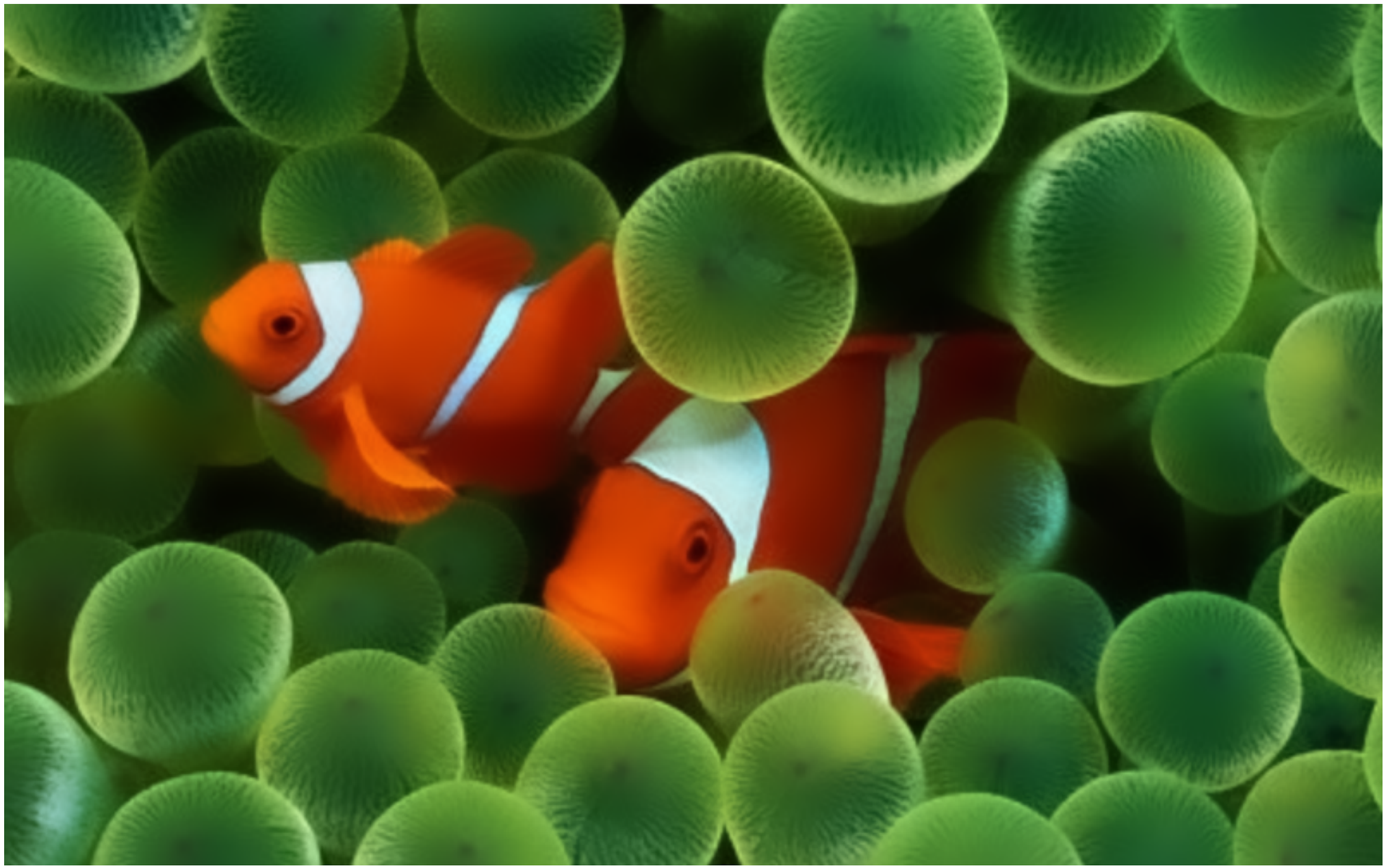
Illustration

Gaussian smoothing

Bilateral Filter

Box filter

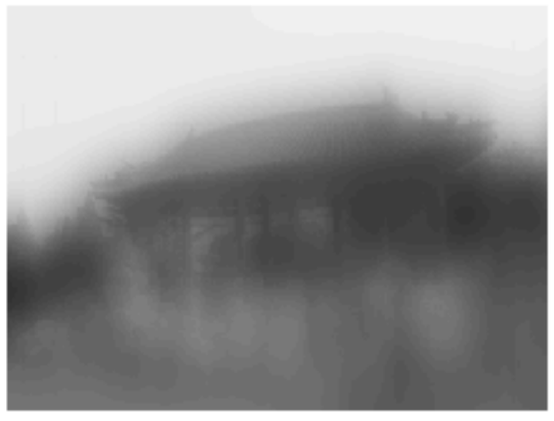
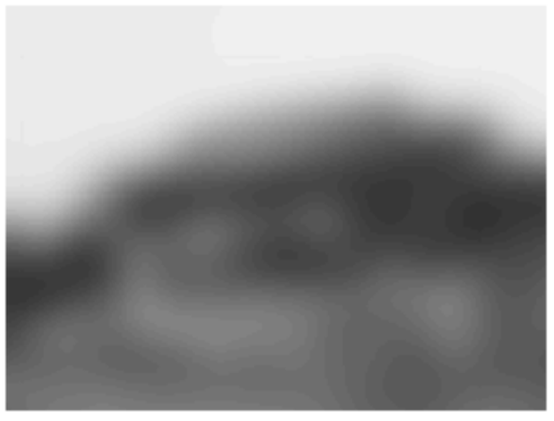
Illustration



Bilateral smoothing

Parameters

Gaussian Filter

σ \ h	0.05	0.2	0.8	∞
4				
8				
16				

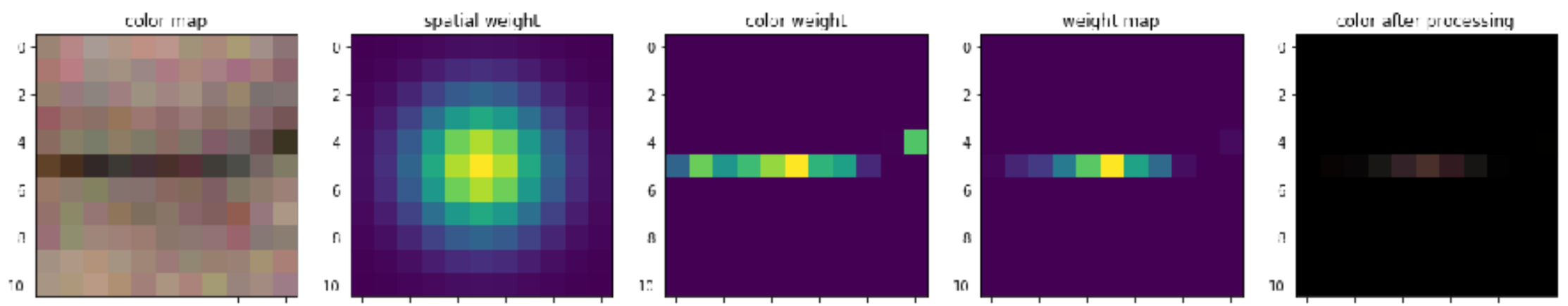
From *A Gentle Introduction to Bilateral Filtering and its Applications*, Durand et al.

Demo Notebook

https://sites.google.com/site/rabinjulien/enseignement

- **demo** using **TP_filtrage_openCV.ipynb** using OpenCV Python Library

- **exercice** with **TP_filtrage_pytorch_exercice.ipynb** using PyTorch



Guided Filtering

- **Principle**: using local, linear regression **on signal values** (ECCV 2010)
- **Advantage**: **much faster**
- **Application**: same as (cross) bilateral filter
- **Formulation**:

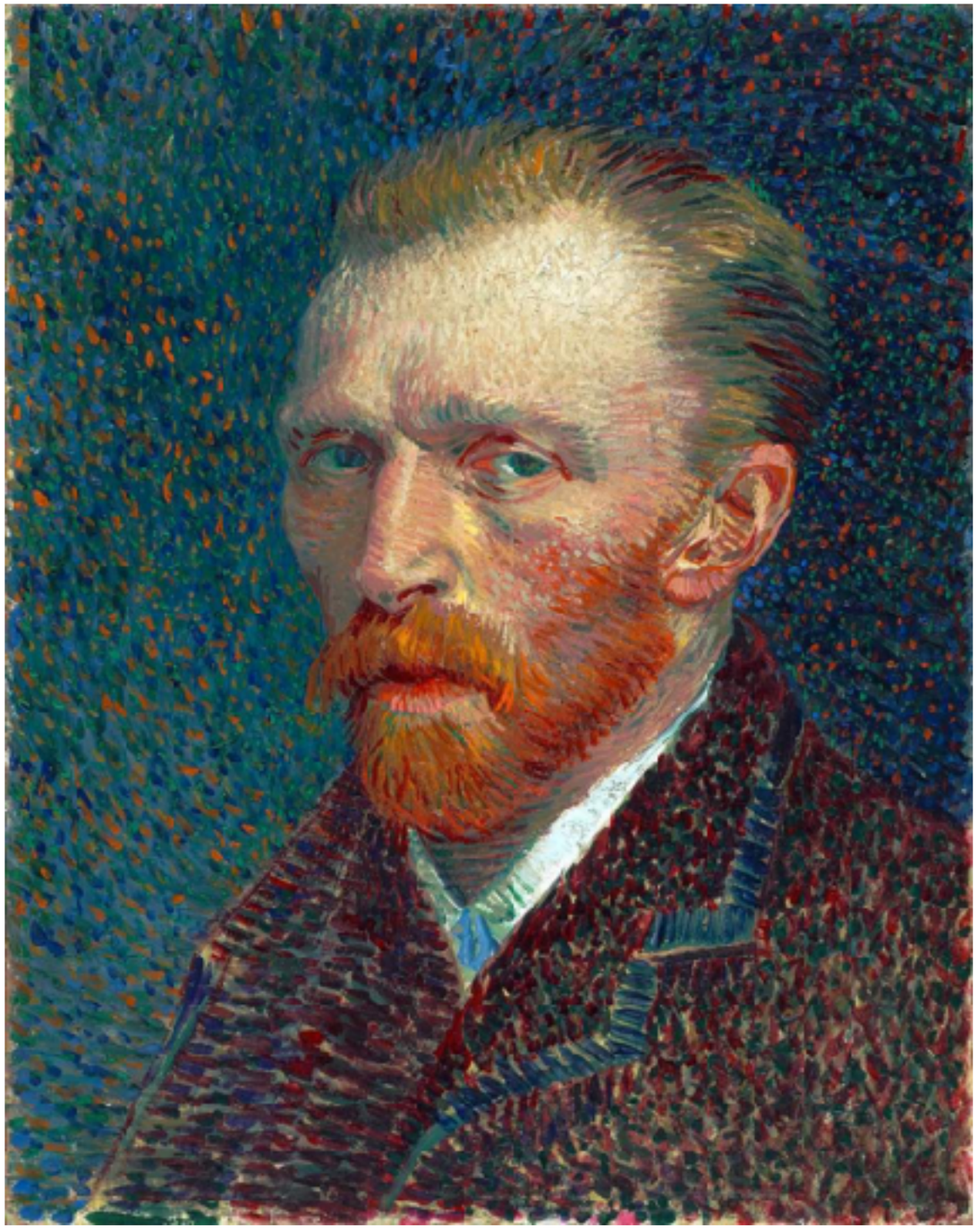
$$\hat{u}(x) = au(x) + b$$

with

$$a = \frac{\hat{\sigma}^2}{\hat{\sigma}^2 + \lambda} \text{ and } b = \hat{\mu}(1 - a)$$

where μ and σ are estimated in the neighborhood (using box filters)

Guided Filtering Illustration



Cross-Bilateral Filter

- Principle: use another image υ to compute color weights
- Application: Multiple exposure/sensing (HDR, Flash, depth...)

$$\hat{u}(x) = \frac{1}{C(x)} \sum_{y \in \Omega} u(y) g_\sigma(y - x) g_h(v(y) - v(x))$$

from *Flash Photography Enhancement via Intrinsic Relighting*, Eisemann & Durand, TOG 2004

(a) photograph with flash
υ

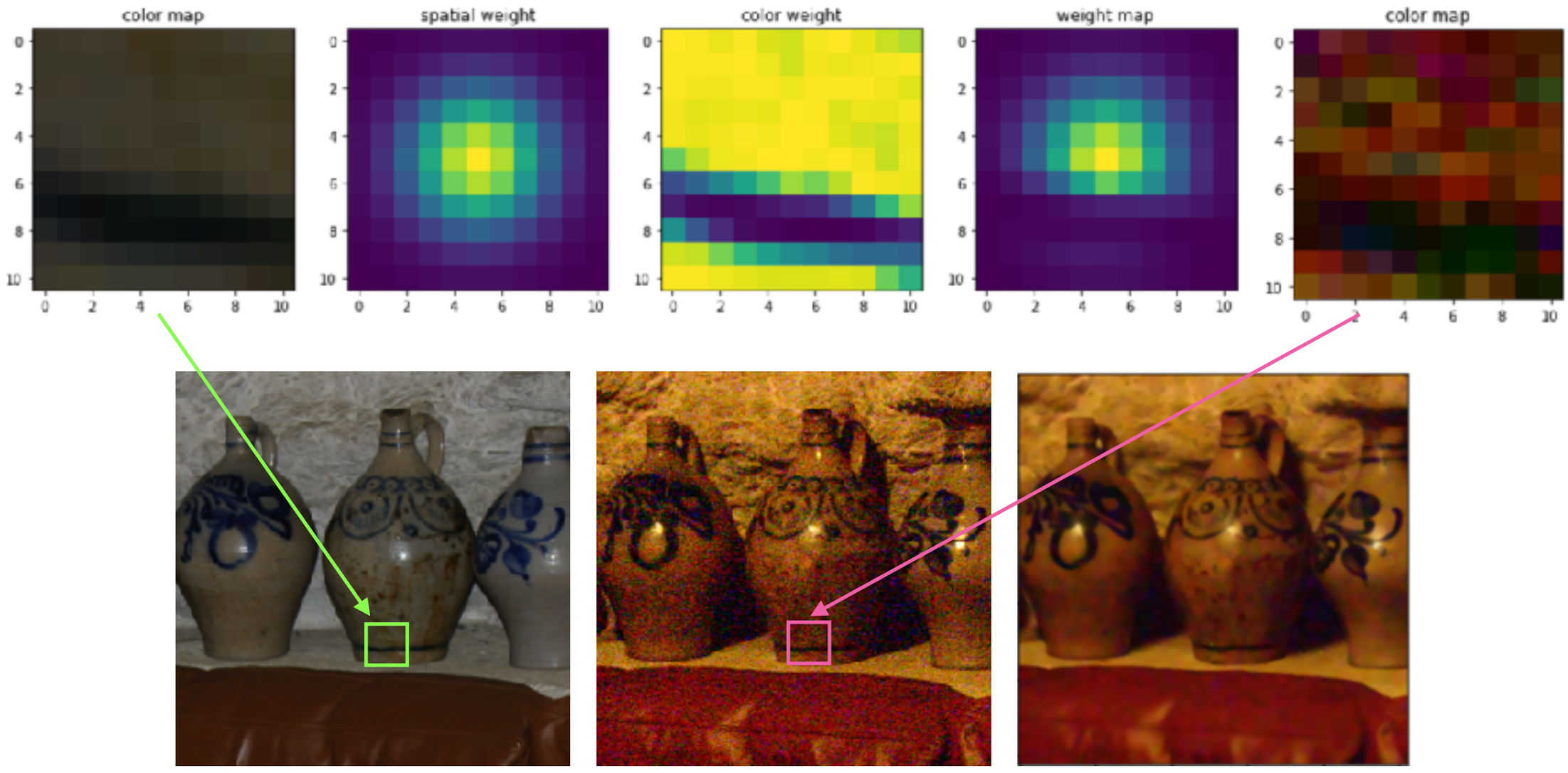
(b) photograph without flash
u

(c) combination
$\hat{u}$ *(w/ HDR blending)*

Demo Notebook

https://sites.google.com/site/rabinjulien/enseignement

- **exercice** with **TP_filtrage_pytorch_exercice.ipynb** using PyTorch



Iterative Filtering

- Principle: **update the color weight** to iterate the filter (Rolling Guidance Filter, ECCV'14)

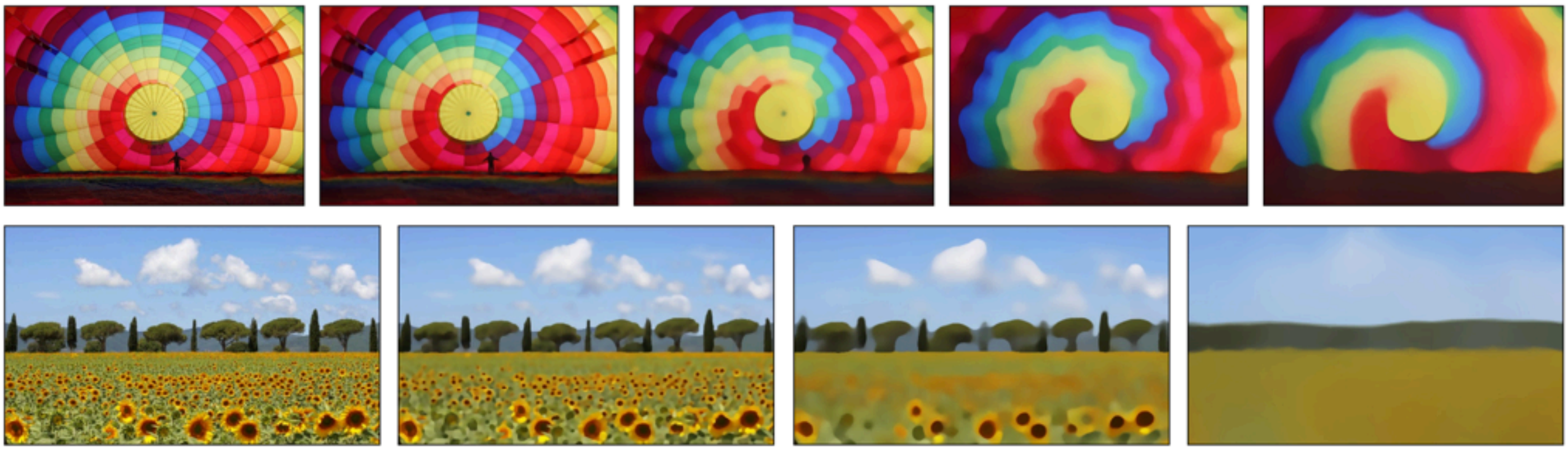
- Application: structure / texture separation

- Formulation with BF:

$$u_{k+1}(x) = \frac{1}{C(x)} \sum_{y \in \Omega} u(y) g_\sigma(y - x) g_h(u_k(y) - u_k(x))$$

- It is different from iterating the filter ! $u_{k+1} \neq \hat{u}_k$

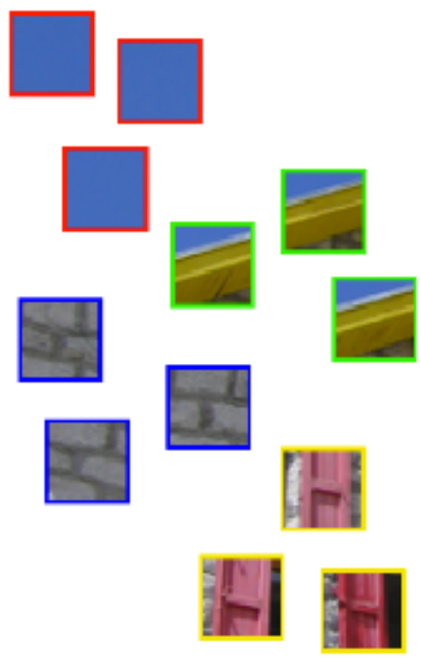
Iterated Filtering Illustration

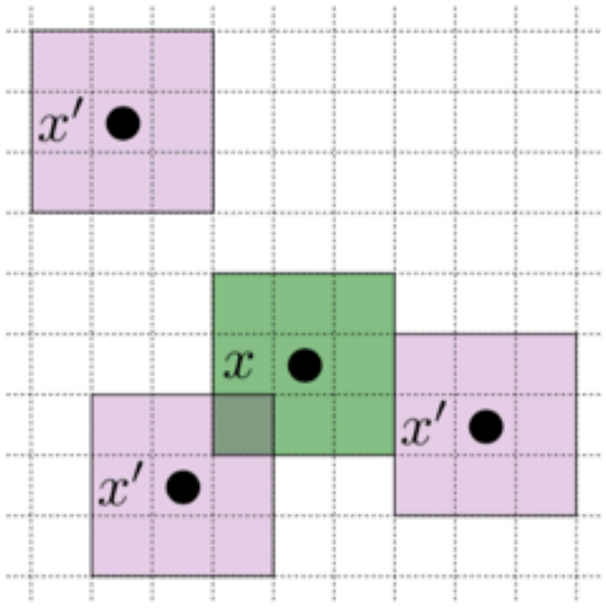


(a) Input (b) BF, 20 times (c) BF, 100 times (d) Upsampling (e) Ours

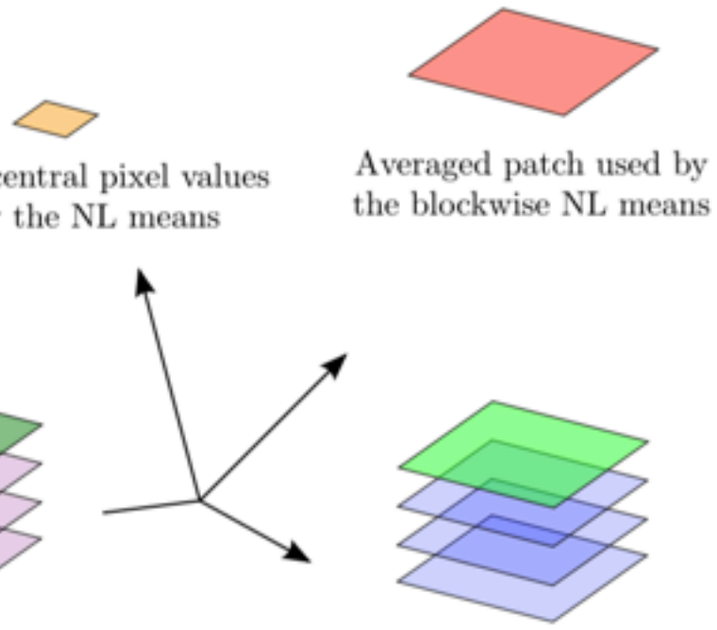
From *Rolling Guidance Filter*, ECCV 2014

Patch-based image processing





Patch extraction



Averaged central pixel values used by the NL means

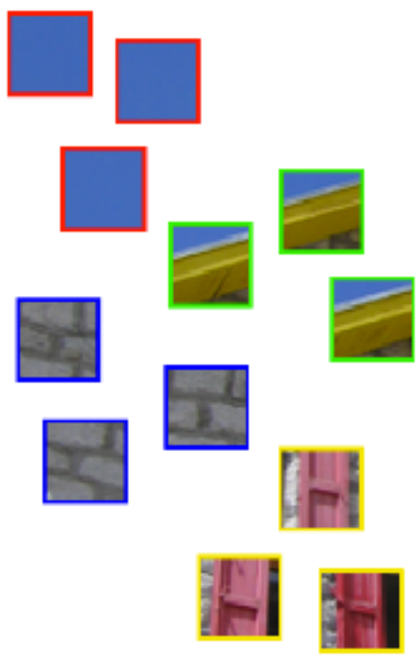
Averaged patch used by the blockwise NL means

Stack of similar noisy patches

Stack of denoised patches after collaborative filtering

Illustrations source: C-A Deledalle & A. Foi

Patch-based image processing



- Patch

  Spatial size $w \times w$

  Dimension $D = d\,w^2$

$$p(x) = (u_i(x+\omega))_{i=1,\dots,d} \in \mathbb{R}^D$$
$$\omega = \{0,\dots,w-1\}^2$$

- Patch distance: $\|p(x) - p(y)\|^2 = \sum_{i=1}^{d} \sum_{t\in\omega} (u_i(x+t) - u_i(y+t))^2$

$$\|p(x) - p(y)\|^2 = \sum_{i=1}^{d} \sum_{t\in\omega} (p_i(x)|_t - p_i(y)|_t)^2$$

Illustrations source: C-A Deledalle & A. Foi

Non-Local means

- **Non-Local Means** [Buades *et al.* 05] (≈bilateral filter for $w = 1$)

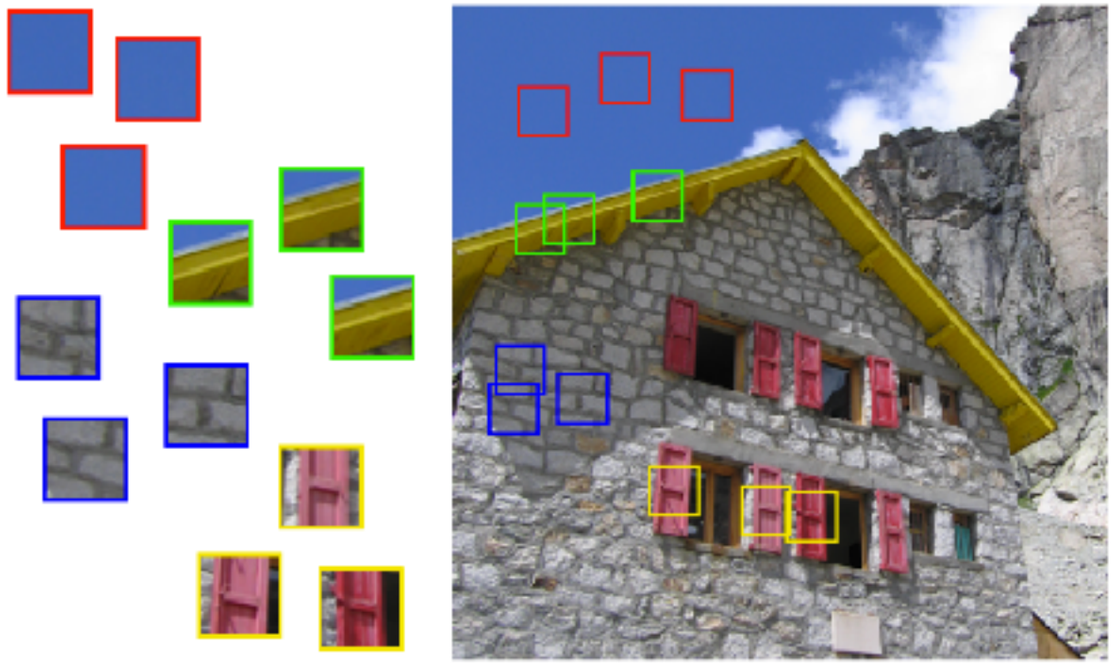
$$\hat{u}(x) = \frac{1}{C(x)} \sum_{y \in W} u(y) e^{-\frac{1}{2\varepsilon^2} \|p(x) - p(y)\|^2}$$

where C is the sum of exponential weights to define a local mean on neighborhood window W

$$C(x) = \sum_{y \in W} e^{-\frac{1}{2\varepsilon^2} \|p(x) - p(y)\|^2}$$

Illustrations source: C-A Deledalle & A. Foi

Example of patch-based image processing: Non-Local means and Non-Local PCA



- **Non-Local PCA** [Deledalle & Salmon'11]: **1)** patch **filtered projection** on PCA basis then **2)** patch **aggregation**

$$\hat{p}(x) = \bar{p} + \sum_{d=1}^{D} \rho\left(\langle p(x) - \bar{p}, v_d \rangle\right) v_d \text{ with } \rho(x) = \begin{cases} x & \text{if } |x| > \tau \\ 0 & \text{otherwise} \end{cases}$$

$$\hat{u}_i(x) = \frac{1}{|\omega|} \sum_{t \in \omega} \hat{p}_i(x - t)|_t$$

Note: patch are centered using patch mean $\bar{p}$

Illustrations source: C-A Deledalle & A. Foi

Exemple: Non-Local means vs Bilateral Filtering

Noisy image u

Denoised image $\hat{u}$

with $w = 1$ (bilateral filter)

with $w = 5$ (Non-Local PCA)

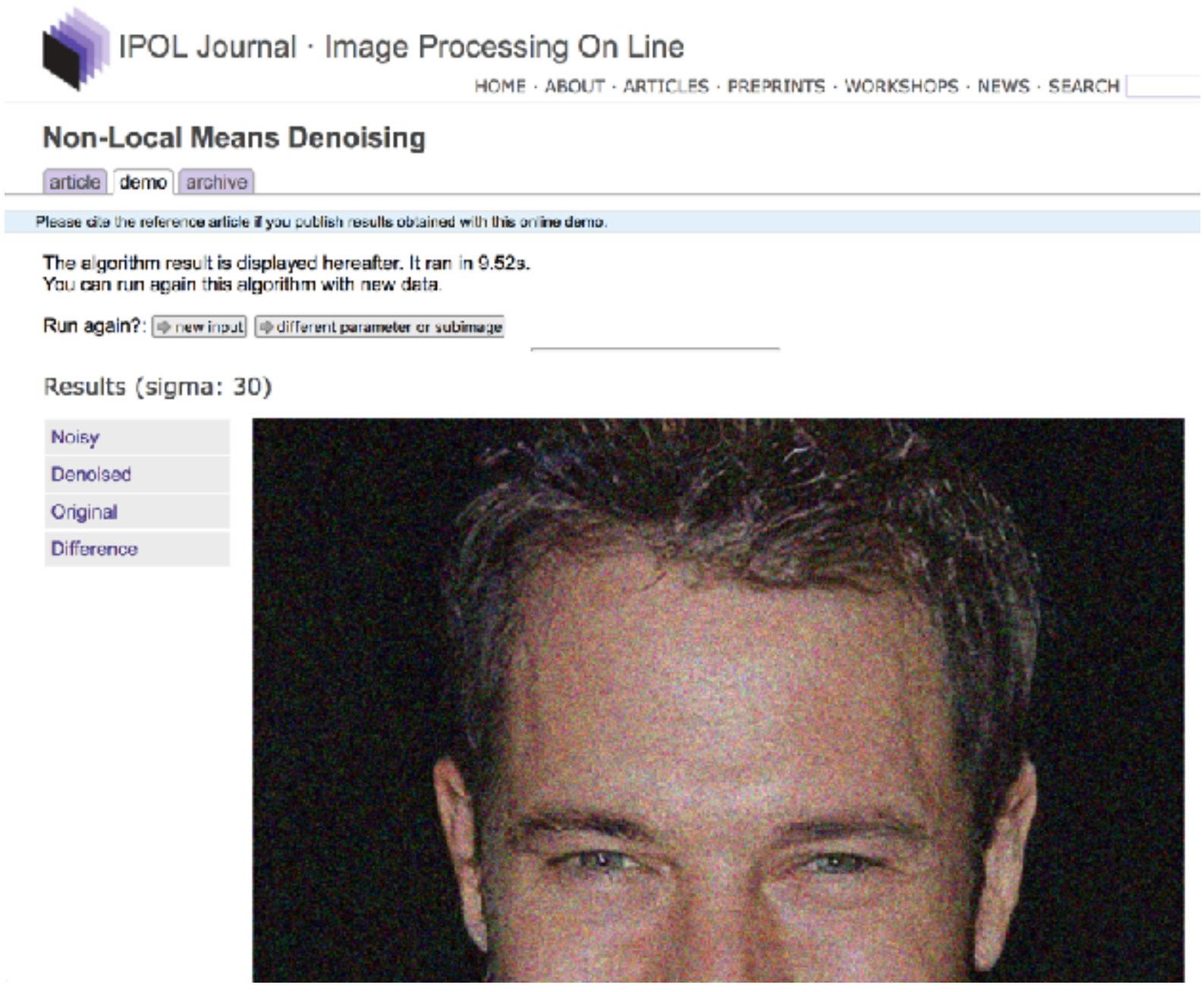
with $w = 5$ (Non-Local mean)

Demo G'M!CoL

https://gmicol.greyc.fr

filter / repair / Smooth [nlmeans]

Demo IPOL



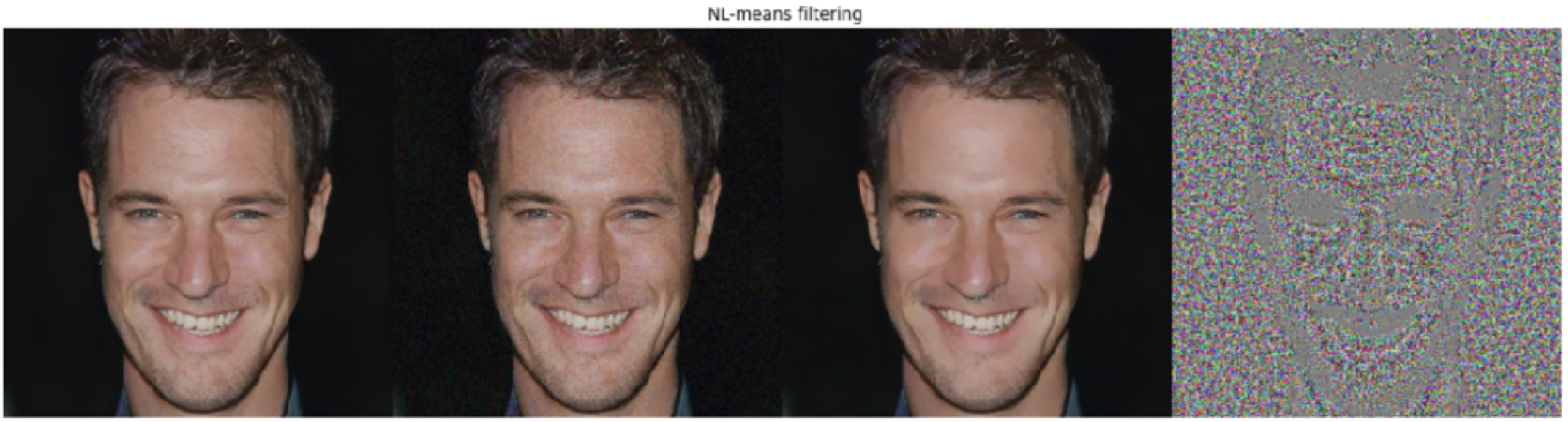
http://demo.ipol.im/demo/bcm_non_local_means_denoising/

Demo Notebook

https://sites.google.com/site/rabinjulien/enseignement

- **demo** : **TP_filtrage_openCV.ipynb** using OpenCV Python Library

- **exercice** : reproduce these results with **TP_filtrage_pytorch.ipynb** using PyTorch Library



Demo Notebook

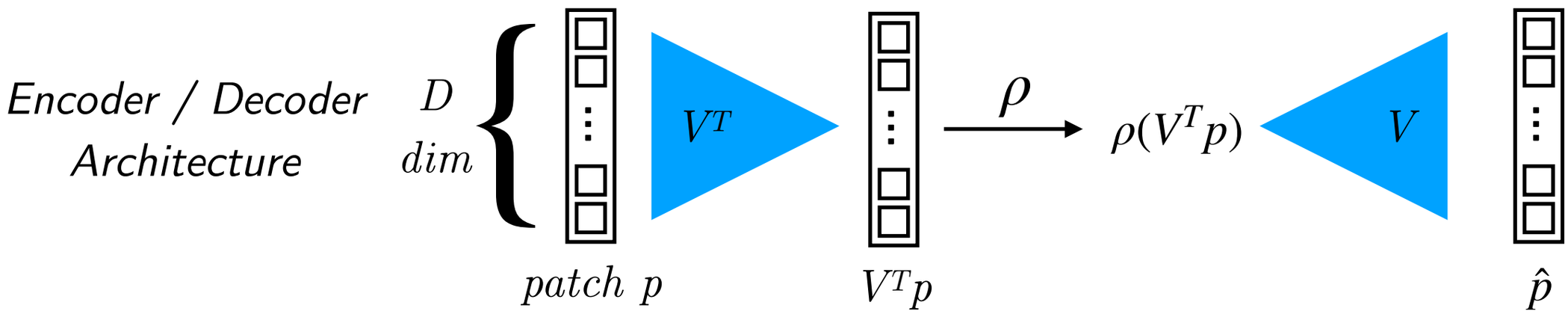
https://sites.google.com/site/rabinjulien/enseignement

- **Matrix notations** : **(see syllabus)**

- **PyTorch notations** :

MLP AE Interpretation

- similar to **MLP** (Multi-linear perceptron): here two linear operations projections using PCA Basis $V = [v_d]_{d=1..D}$ with non-linear ρ (symmetric ReLU)



- **Non-Local PCA**: **1)** patch **filtered projection** on PCA basis then **2)** patch **aggregation**

$$\hat{p}(x) = \bar{p} + \sum_{d=1}^{D} \rho\left(\langle p(x) - \bar{p}, v_d \rangle\right) v_d \text{ with } \rho(x) = \begin{cases} x & \text{if } |x| > \tau \\ 0 & \text{otherwise} \end{cases}$$

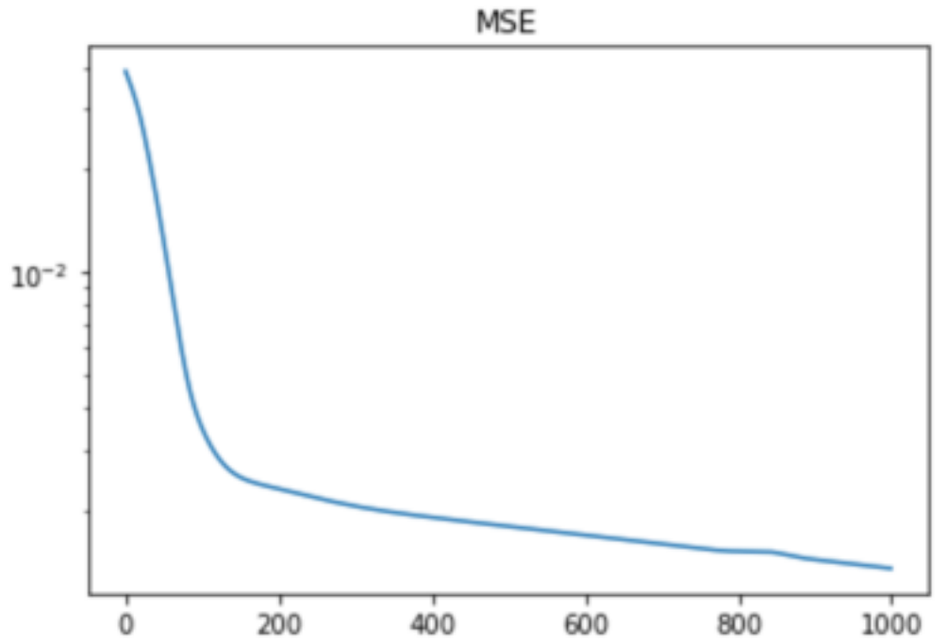
$$\hat{u}_i(x) = \frac{1}{|\omega|} \sum_{t \in \omega} \hat{p}_i(x - t)|_t$$

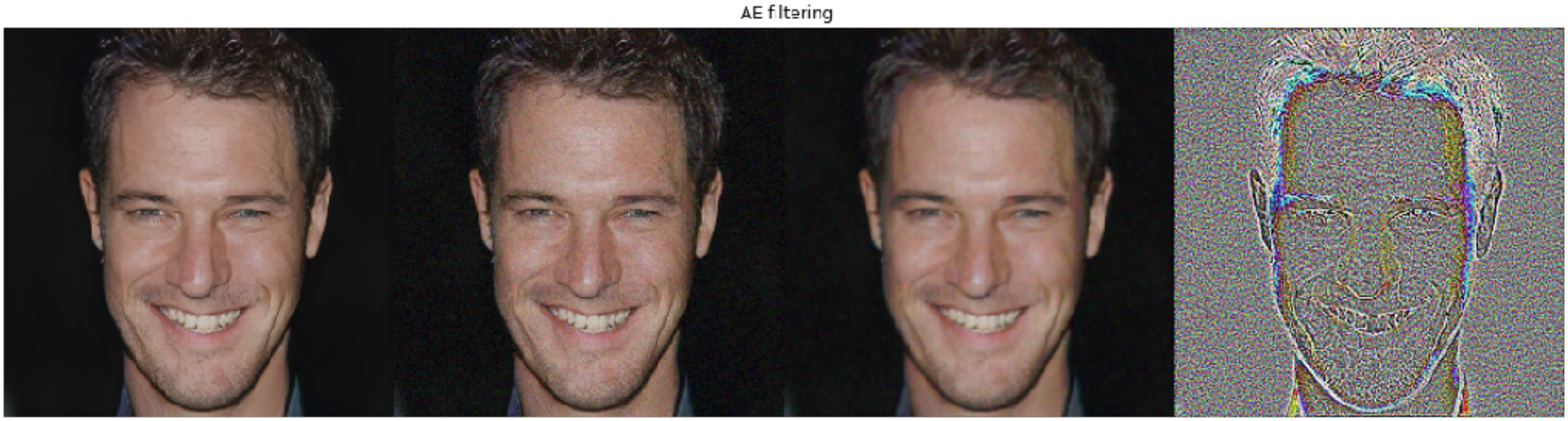
Demo Notebook

https://sites.google.com/site/rabinjulien/enseignement

Demo : training an patch-based AE

TP_filtrage_pytorch.ipynb





Attention Interpretation

« Attention Is All You Need », Vaswani, NIPS'17

Attention Mechanism

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d}}\right)V$$

where d is the embedding dimension

- Q : Query = input (encoded)
- K : Keys = memory / codebook
- V : Values = output (encoded)

and $\quad \text{softmax}(z)|_i = \frac{e^{z_i}}{\sum_j e^{z_j}}$

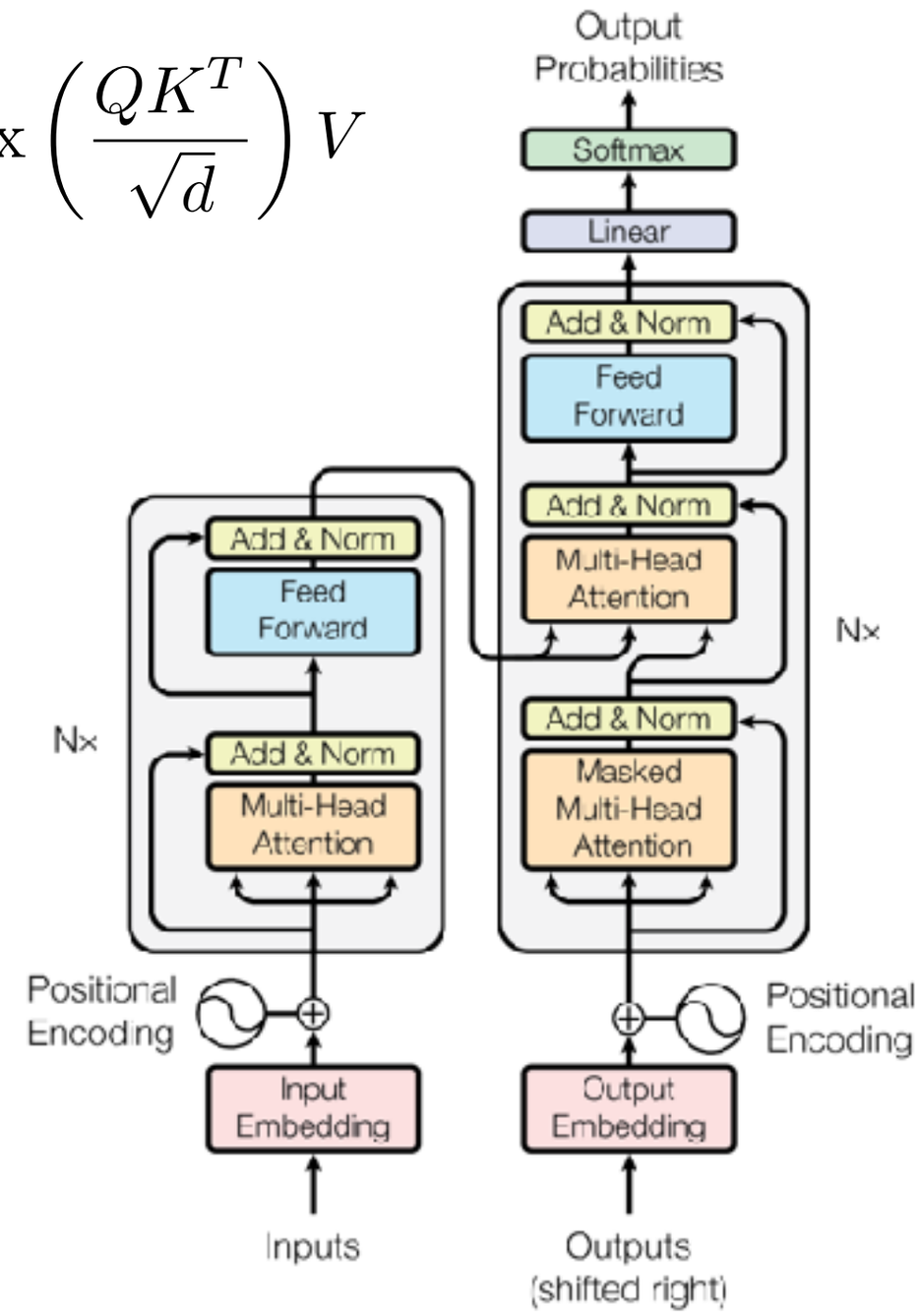
Note the scaling factor depending on d

Self-Attention : Q = V = K

Cross-Attention Mechanism : Q & K=V

Positional Embedding

Multi-head Attention : embeddings Wk, Wq and Wv



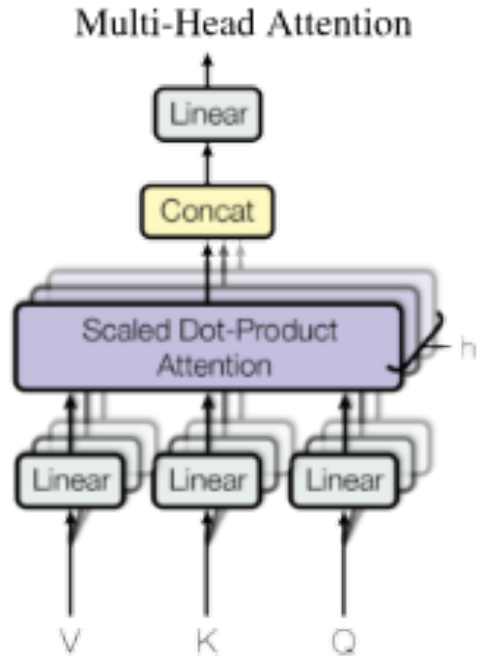
Transformer Architecture

Attention Interpretation

Dot-product Self-Attention (SA) $z \in \mathbb{R}^{N \times d}$

$$[q, k, v] = zU_{QKV} = z[U_Q, U_K, U_V]$$

$$\forall i \in 1..N,\ \hat{q}_i = SA(q, k, v)|_i = \frac{1}{\sum_j e^{\frac{q_i . k_j}{\sqrt{d}}}} \sum_j e^{\frac{q_i . k_j}{\sqrt{d}}} v_j$$



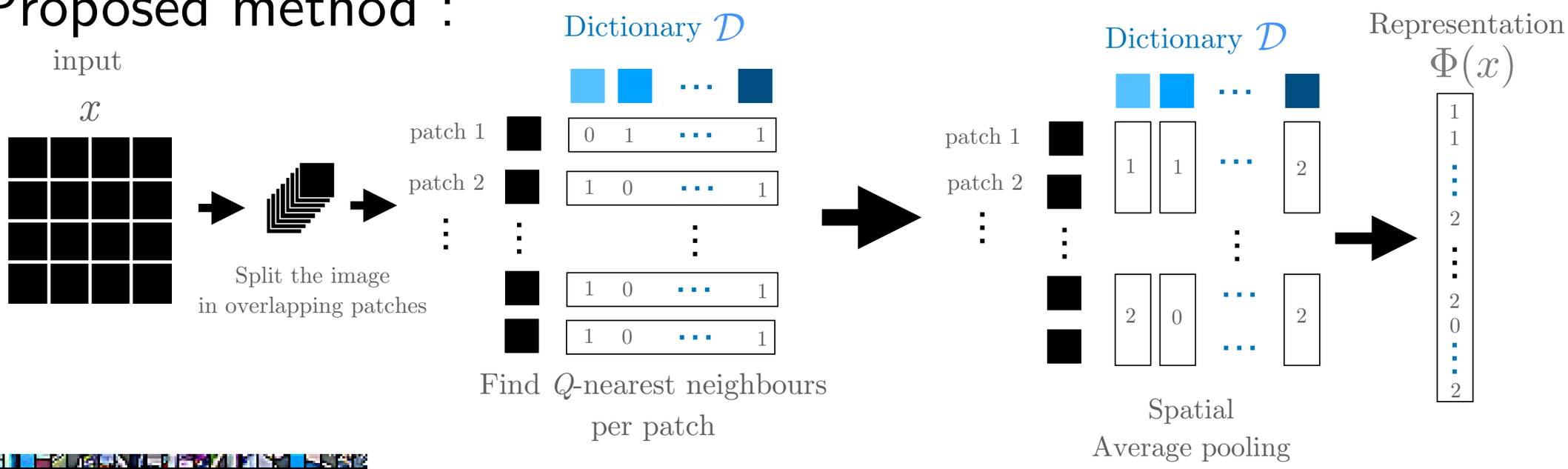
NL-means on input z with PCA $[q, k, v] = z[U_Q, U_Q, U_V]$

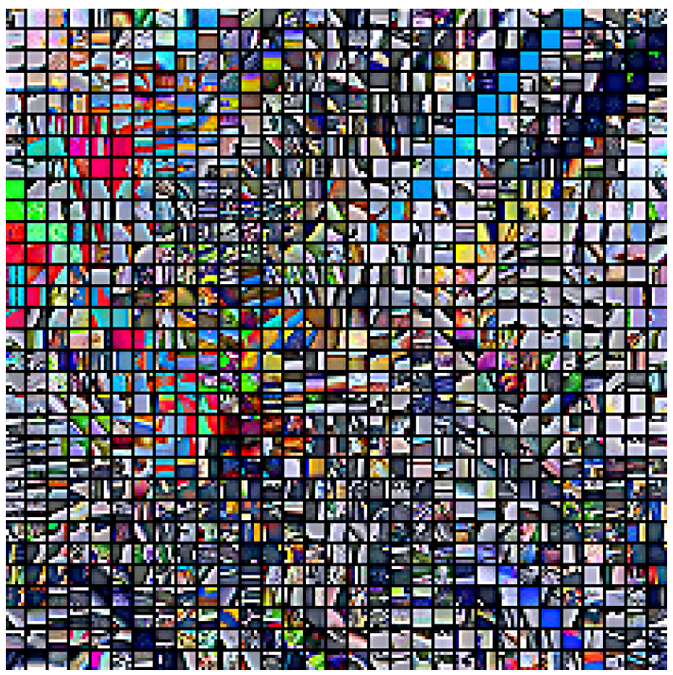
$$\forall i \in \{1..N\},\ \hat{q}_i = \frac{1}{\sum_j e^{-\frac{\| q_i - k_j \|^2}{h}}} \sum_j e^{-\frac{\| q_i - k_j \|^2}{h}} v_j$$

$$\hat{q}_i \propto \sum_j e^{\frac{2 q_i . k_j - \| q_i \|^2 - \| k_j \|^2}{h}} v_j$$

Are patch useful for Deep Learning ?

- **Example 1**: "The Unreasonable Effectiveness of Patches in Deep Convolutional Kernels Methods " [Thiry ICML'21]
- Proposed method :



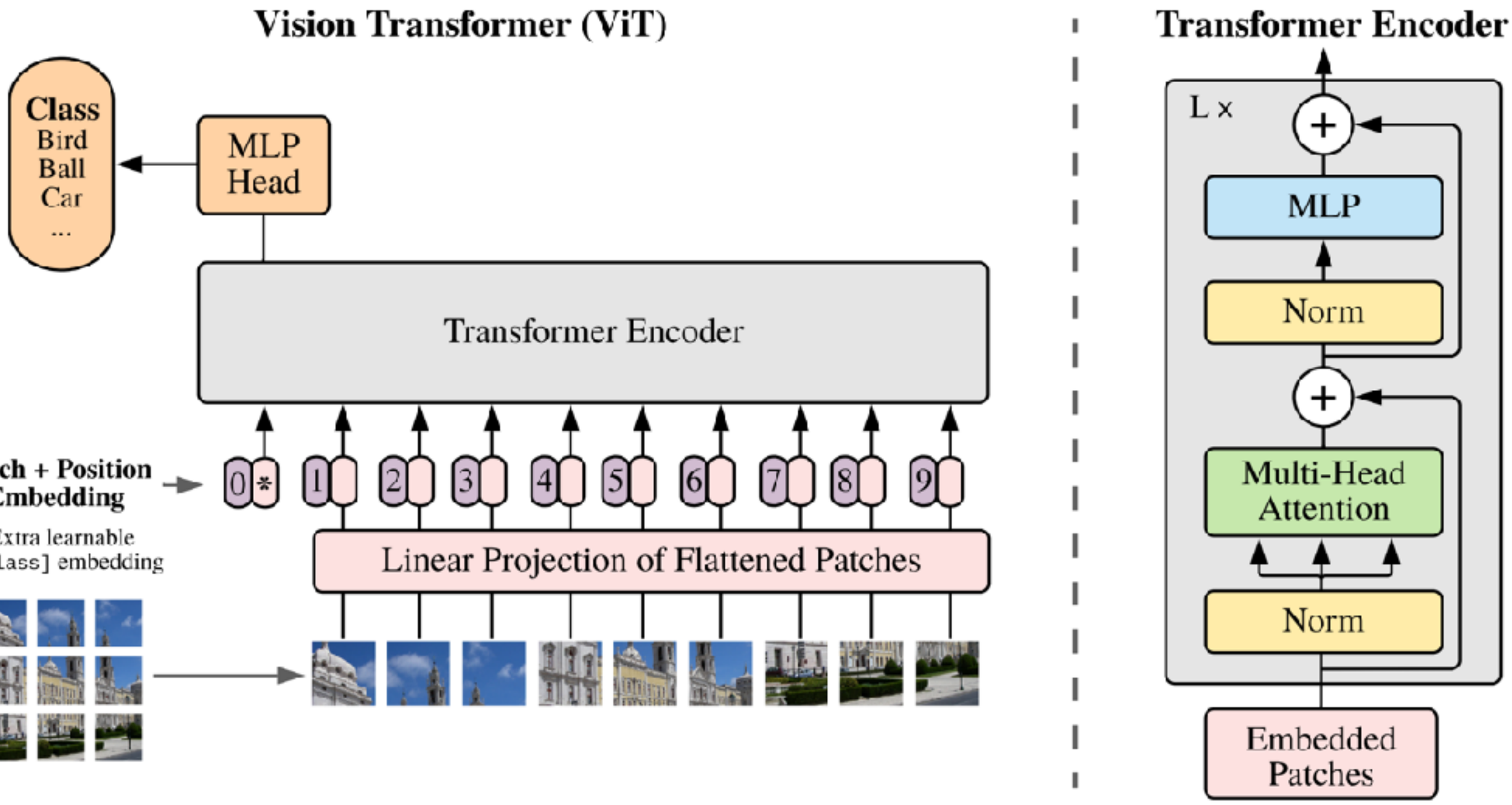


- Result on ImageNet : with 2000 random 12x12 patches for 64x64 images :
  - Random Guess : Top 1 0.1 %
  - Random CNN + Linear Classifier : 19%
  - Patch + Linear Classifier : 36%
  - VGG-19 : 74.5%

Are patch useful for Deep Learning ?

- Example 2: **Vision Transformers**

Source: « An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale », ICLR'21



What is next ?

Acceleration of patch search

Applications of **patch** in image processing and computer vision (beyond image filtering)

- Image **Editing** (inpainting, retargeting, seam carving, ...)
- Image **Synthesis** (image reshuffling, texture synthesis)
- Style **Transfer** (image analogies, image stylization)

Interactive Image Editing

Image matching

Most computer vision algorithms for image **matching** are based on the comparison of local features (*e.g.* **patch**).

- **Applications:** image stitching (panorama), image registration, template matching, indexation, classification**,** stereo-vision (dense correspondance) ...
- **Tools:**

  - Feature extraction: here simply **patch**, but can be more involved (invariant / robust / salient) e.g. SIFT [Lowe, 2004]
  - Matching step: **Nearest Neighbor** (NN) search.

Example: SIFT matching

Object detection & recognition by finding local correspondances

Example: image registration

Image registration using projective geometry (homography) and geometric consensus (RANSAC)

Demo IPOL

IPOL Journal · Image Processing On Line

HOME · ABOUT · ARTICLES · PREPRINTS · WORKSHOPS · NEWS · SEARCH

Fast Affine Invariant Image Matching

Mariano Rodríguez, Julie Delon, Jean-Michel Morel

article | demo | archive

published • 2018-09-24

reference • Mariano Rodríguez, Julie Delon, and Jean-Michel Morel, *Fast Affine Invariant Image Matching*, Image Processing On Line, 8 (2018), pp. 251–281. https://doi.org/10.5201/ipol.2018.225

BibTeX info

Communicated by Pablo Musé
Demo edited by Mariano Rodríguez

Abstract

Methods performing Image Matching by Affine Simulation (IMAS) attain affine invariance by applying a finite set of affine transforms to the images before comparing them with a Scale Invariant Image Matching (SIIM) method like SIFT or SURF. We describe here how to optimize IMAS methods. First, we detail an algorithm computing a minimal discrete set of affine transforms to be applied to each image before comparison. It yields a full practical affine invariance at the lowest computational cost. The matching complexity of current IMAS algorithms is

Fast Affine Invariant Image Matching https://www.ipol.im/pub/art/2018/225/

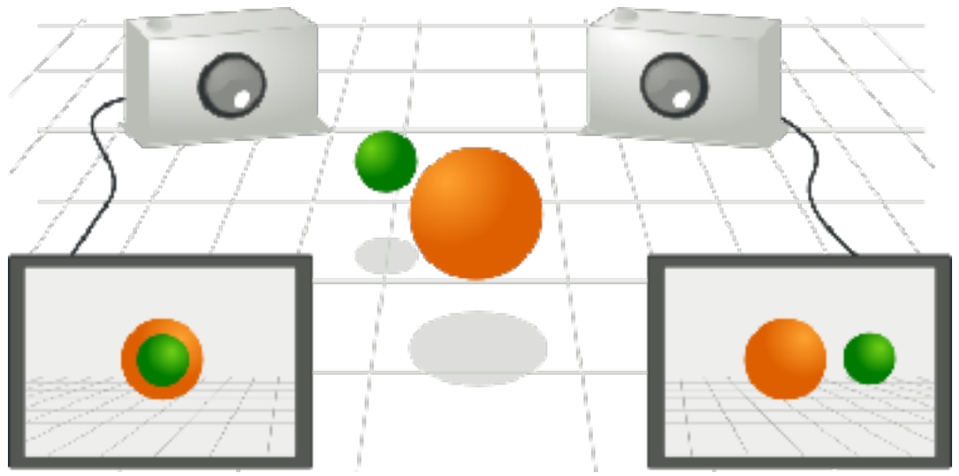
Example: image stitching

Image registration (non-rigid lens deformation)

Blending images with color correction [Brown & Lowe, 2007]

Illustrations source: Brown

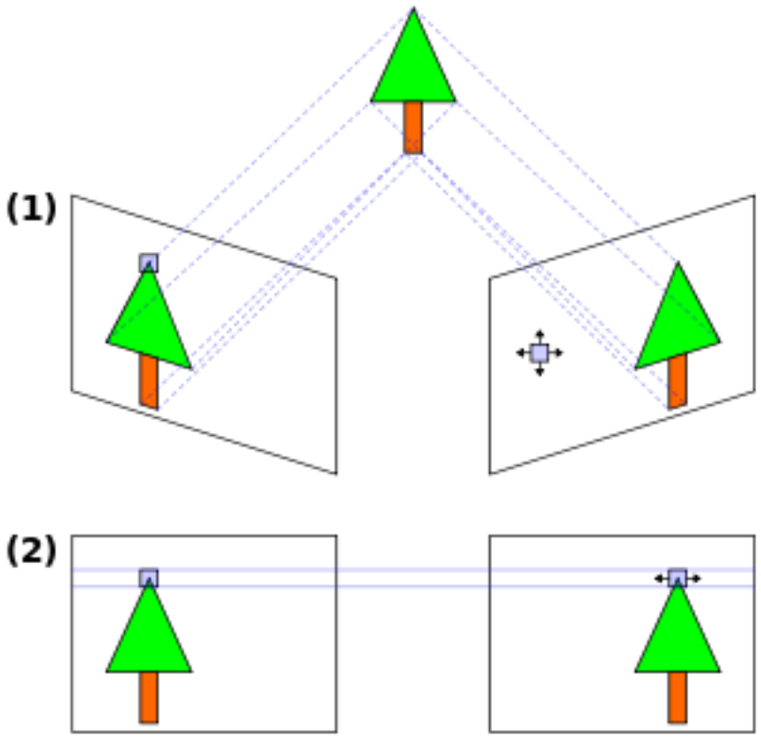
Example: Stereo-Vision

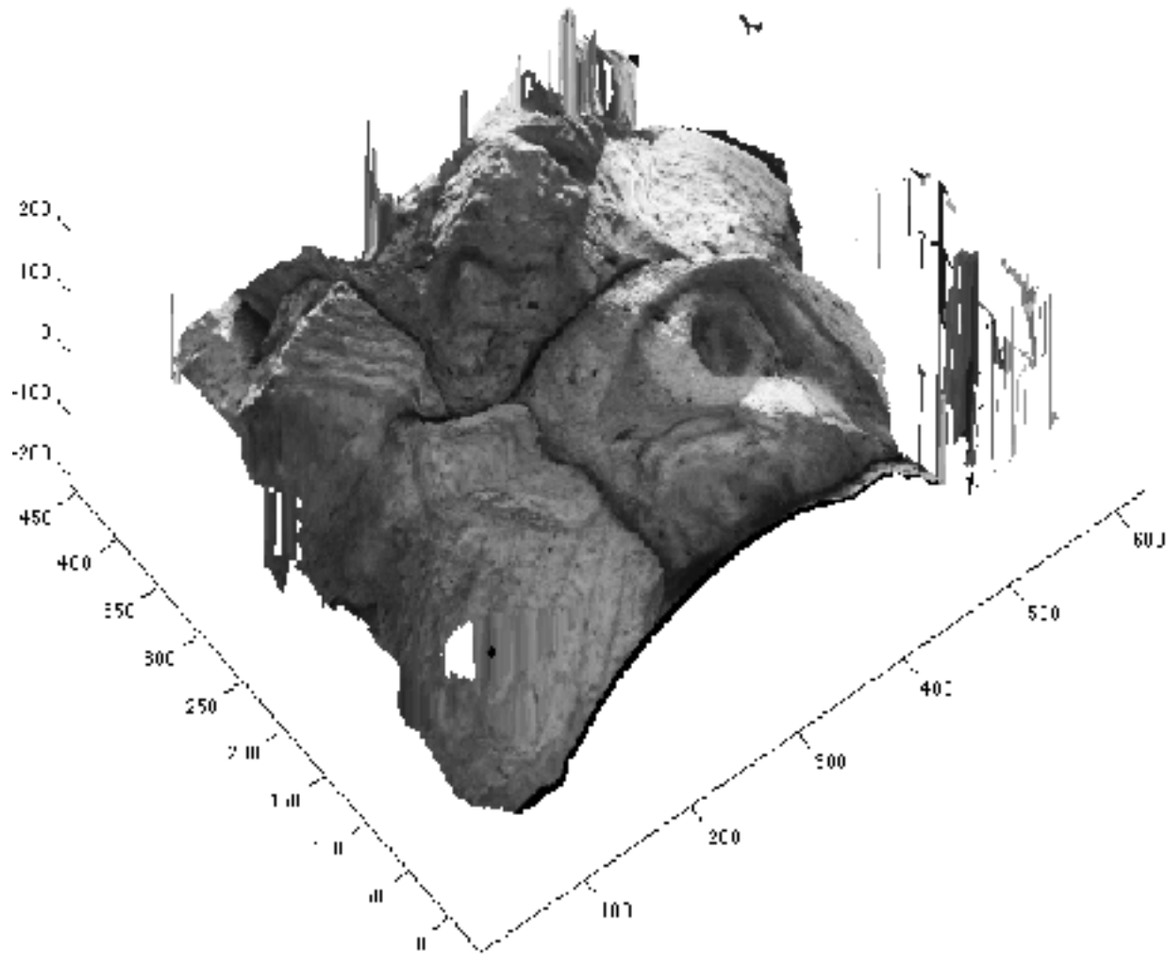


Setting: two cameras filming the same scene under 2 different viewpoints (like human vision)

Illustrations source: WikiCommons and P. Monasse

Example: Stereo-Vision





After image rectification (using epipolar geometry) and **dense matching**, **depth** can be simply estimated from the displacement between images

NN is computationally intensive !

- Patches live in **high dimensions** (curse of dimensionality)
  - *e.g.* 8x8 color patches dimension is 192
- Exhaustive search is **intractable**
  - *e.g.* comparing all patches from two HD images ($1{,}920 \times 1{,}080 \approx 2.10^6$ pixels) requires 4.10^{12} comparisons !
- NN has to be **iterated** to incorporate constraints
- **Approximations and tricks are needed** (dimension reduction, quantization, data structure, approximate search, parallel computing, etc.)

PatchMatch Overview

- PatchMatch [Barnes *et al.*, 2009] is a fast algorithm to estimate dense **NN field** between images, using several heuristics:

- **Initialization:** random correspondance between images (few of which being correct by pure chance)

- **Iteratively (~4 iterations):** try improving correspondances by **sequentially** updating pixels of the NN field in **scan order** (from left to right, top to bottom) on odd iteration (and reverse scan order on even iteration)

    1. Propagation **from the two nearest neighbors**

    2. Sparse random search in the neighborhood of the current candidate

- **Improvements**: multi-scale, k-NN, metric, patch rotation and symmetry ... *Generalized PatchMatch* [Barnes *et al.*'10], *Image melding* [Darabi *et al.*'12]

PatchMatch: A Randomized Correspondence Algorithm for Structural Image Editing

Connelly Barnes[1], Eli Shechtman[2,3],
Adam Finkelstein[1], and Dan B Goldman[2]

[1]Princeton University
[2]Adobe Systems
[3]University of Washington

PatchMatch: details

- Let A be the image to be edited, using patches from image B.
  One want to find the **NN field** f (or *position map)* to reconstruct image A, s.t.

$$\hat{A}(x) = B(f(x)) \simeq A(x)$$

An optimal NN field is defined as: $$f(x) = \arg\min_y D(y) := \|p_A(x) - p_B(y)\|$$

Remark: *Other choice than Euclidean distance can be used (such as L_1 distance, Lab color space, gradient field ...)*

- Due to regularity/smoothness in natural images (flat areas, shadings, borders, etc.), it is very likely that around each location x (using a small vector t)

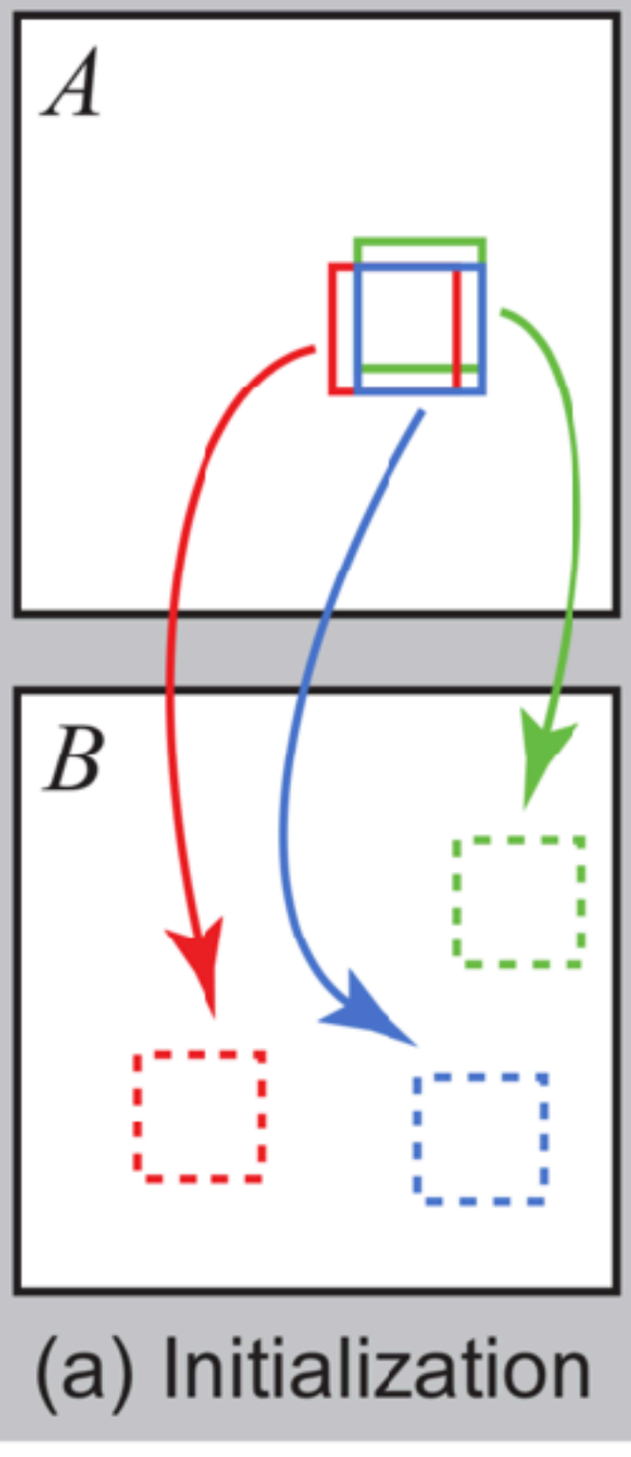
$$f(x+t) = f(x) + t \quad (\text{i.e. } f(x) = f(x+t) - t)$$

Remark: *In [Barnes et al., 2009], A and B have the same size so that the NN field f is defined from an offset map o, i.e.*

$$f(x) = x + o(x) \qquad \hat{A}(x) = B(x + o(x))$$

so that the offset map is ***piece wise constant****: a.e. we have* $$o(x+t) = o(x)$$

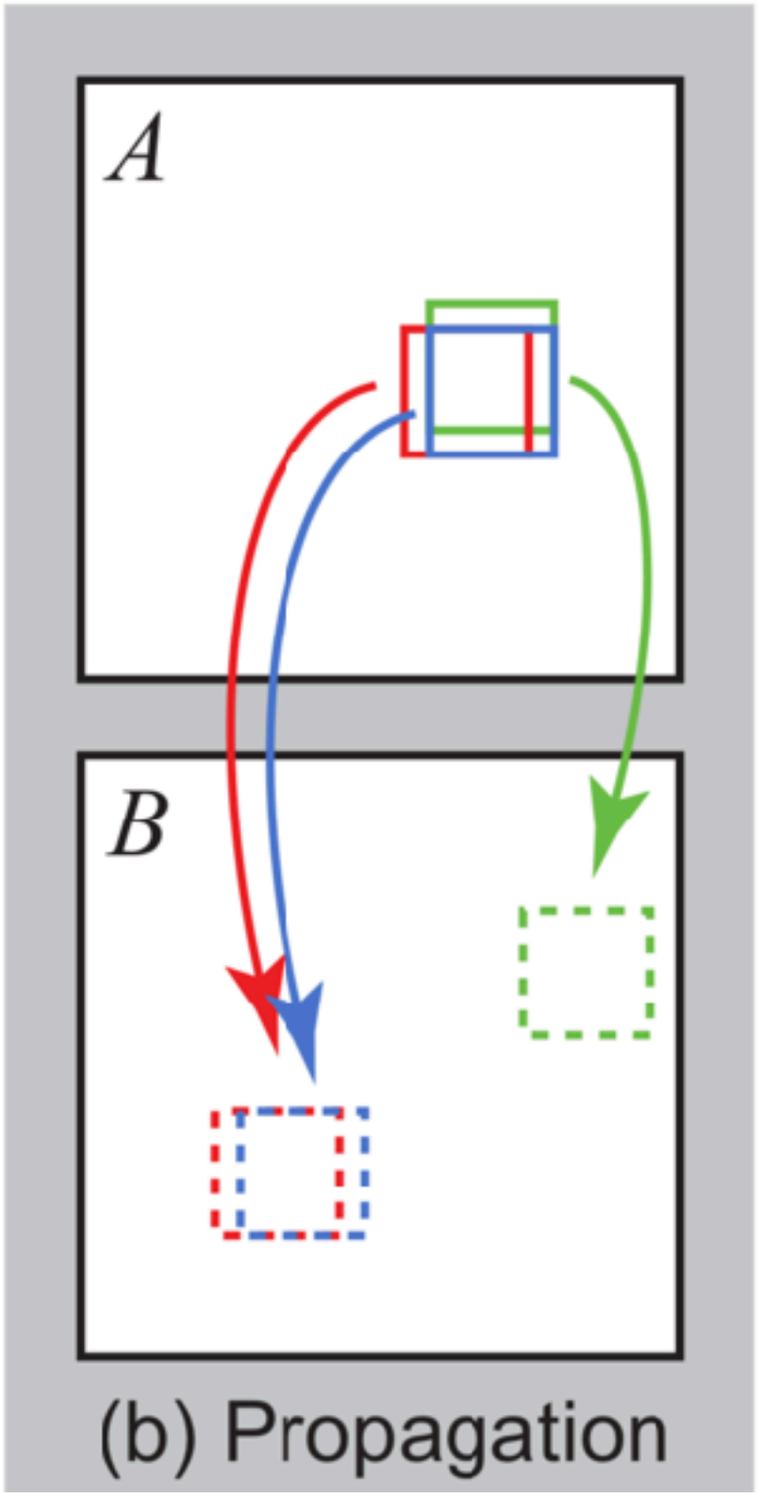
PatchMatch: details



- Algorithm start with **random initialization**
- ... or estimated from a coarser scale, when using coarse to fine approach
- The offset map is updated **sequentially**, where each pixel is updated in scan order (then, in opposite order)
- Each update is a **propagation** step (2 comparaisons) followed by a **random search** (k comparisons)
- Number of comparison is $\mathrm{O}(5(2+\mathrm{k})\mathrm{N})$, where N is the number of pixels (instead of $\mathrm{O}(\mathrm{N}^2)$).
- Patch size $w=7$

Illustrations source: Barnes

PatchMatch: details



- **Propagation** step reads as follows:
  Offset map o and position map f are **updated** sequentially, by considering for each pixel x two neighbors a and b

  - for **odd** iteration, pixels x of image A are read in scanline order (*i.e.* left to right, from top to bottom)
    $\mathrm{a}=(0,\text{-}1)$ (left) and $\mathrm{b}=(0,\text{-}1)$ (top)

  - for **even** iteration, pixels x of image A are read in **reverse** scanline order (*i.e.* right to left, from bottom to top)
    $\mathrm{a}=(0,+1)$ (right) and $\mathrm{b}=(0,+1)$ (bottom)

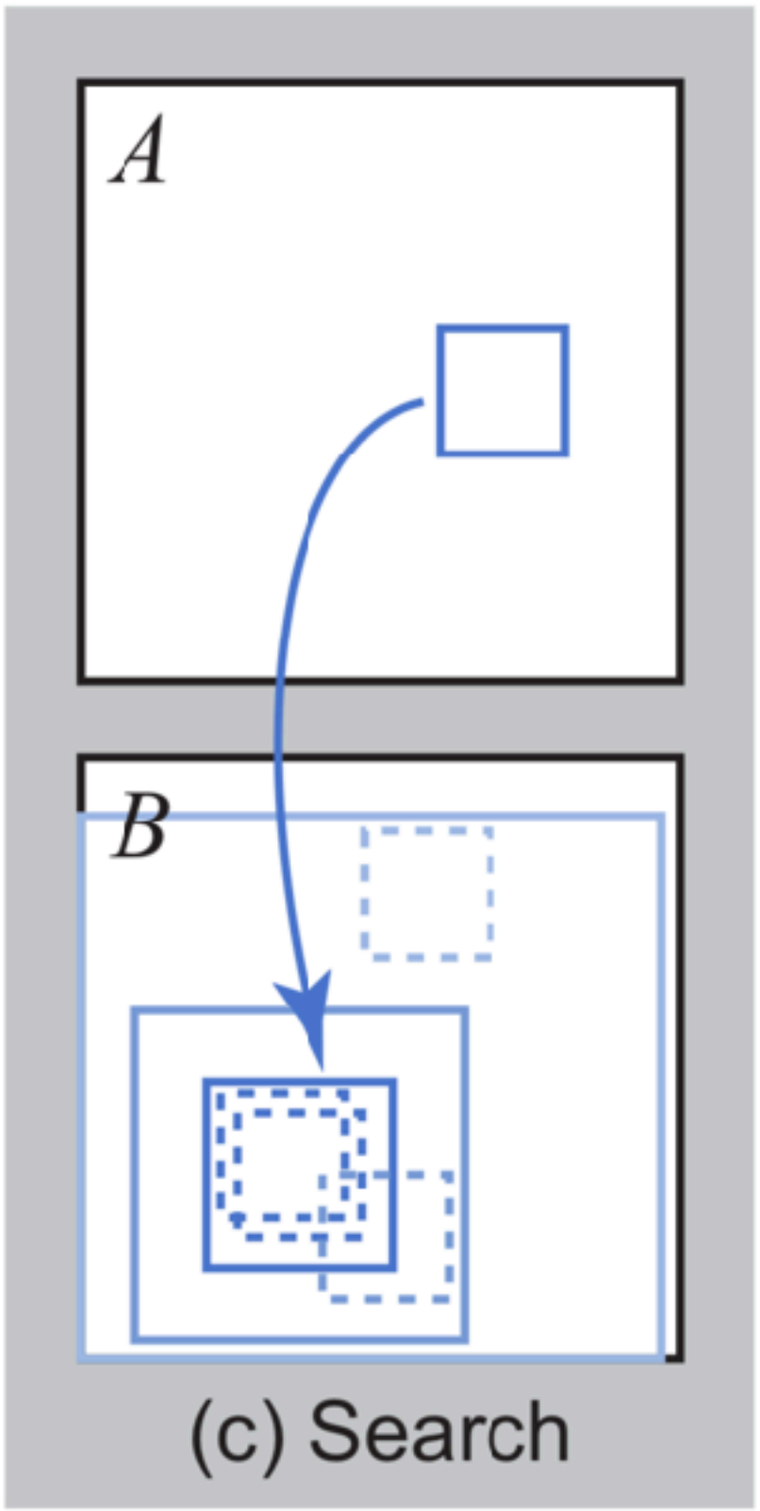
- **Update rule**

$$o(x) \leftarrow \arg\min_{y} D(x+t) \text{ s.t. } t \in \{o(x), o(x+a), o(x+b)\}$$

$$\text{where} \quad D(y) = \|p_B(y) - p_A(x)\|$$

Illustrations source: Barnes

PatchMatch: details



- **Random search** step reads as follows:

  For pixel x, **draw random patch candidates** y_i in the neighborhood of the current NN position $y_0 = f(x)$ (i.e. with offset $t_0 = o(x)$)

  - different neighborhood (centered on y_0) are defined, with decreasing radius $r_i = \alpha^i R$ until $r_i < 1$ pixel
  - In practice: $\alpha = 1/2$, and R is the radius of image B
  - Random candidates $y_i = y_0 + r_i X_i$ (or offsets $t_i = t_0 + r_i X_i$) with random variables $X_i \sim U([-1,1]^2)$

- **Update rule**

$$o(x) \leftarrow \arg\min_y D(x+t) \text{ s.t. } t \in \{t_0, t_1, \ldots, t_i, \ldots\}$$

where $D(y) = \|p_B(y) - p_A(x)\|$

Illustrations source: Barnes

Demo Notebook

https://sites.google.com/site/rabinjulien/enseignement

PatchMatch_Exercice.ipynb

Texture Synthesis

Texture synthesis by example

- **Goal**: from an example image, synthesize a new image which is perceptually similar

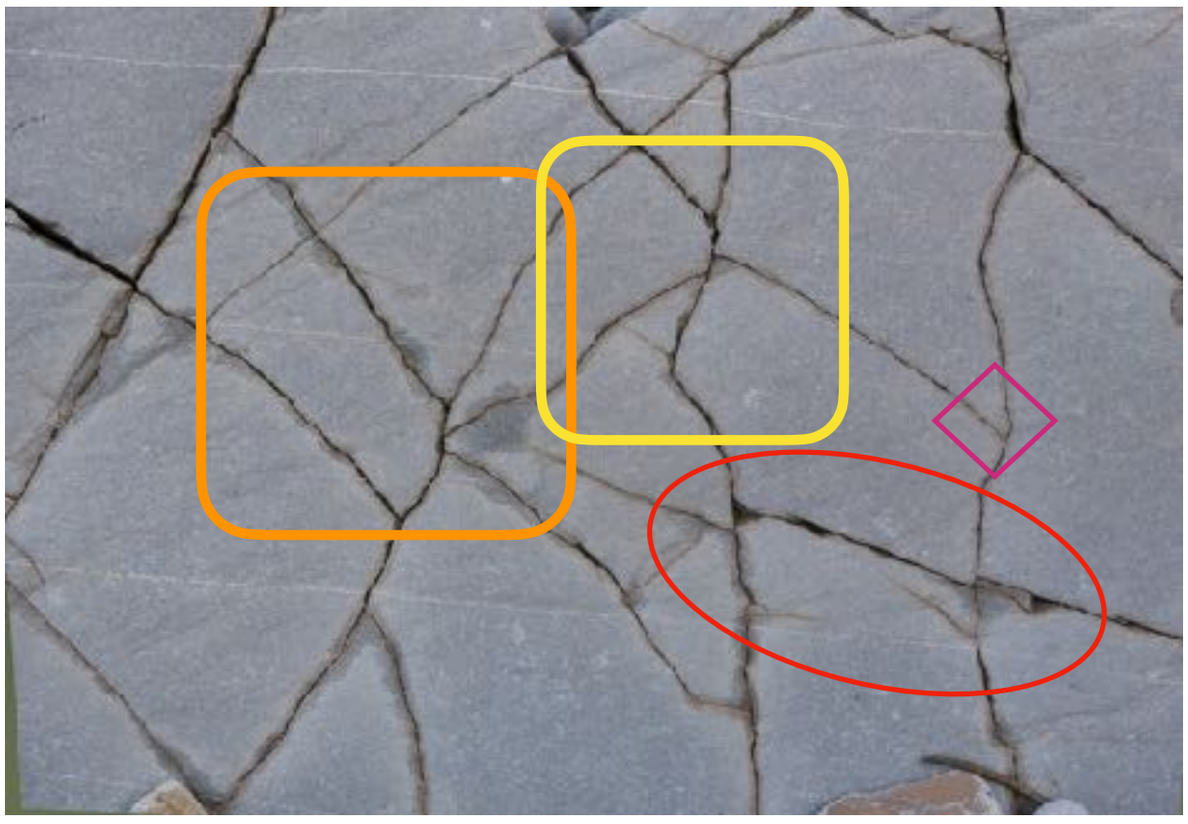
Example

Ideal synthesis

- **Hypothesis**: input image is stationary

Synthesis by reshuffling

- **Idea**: using PatchMatch to resample the image



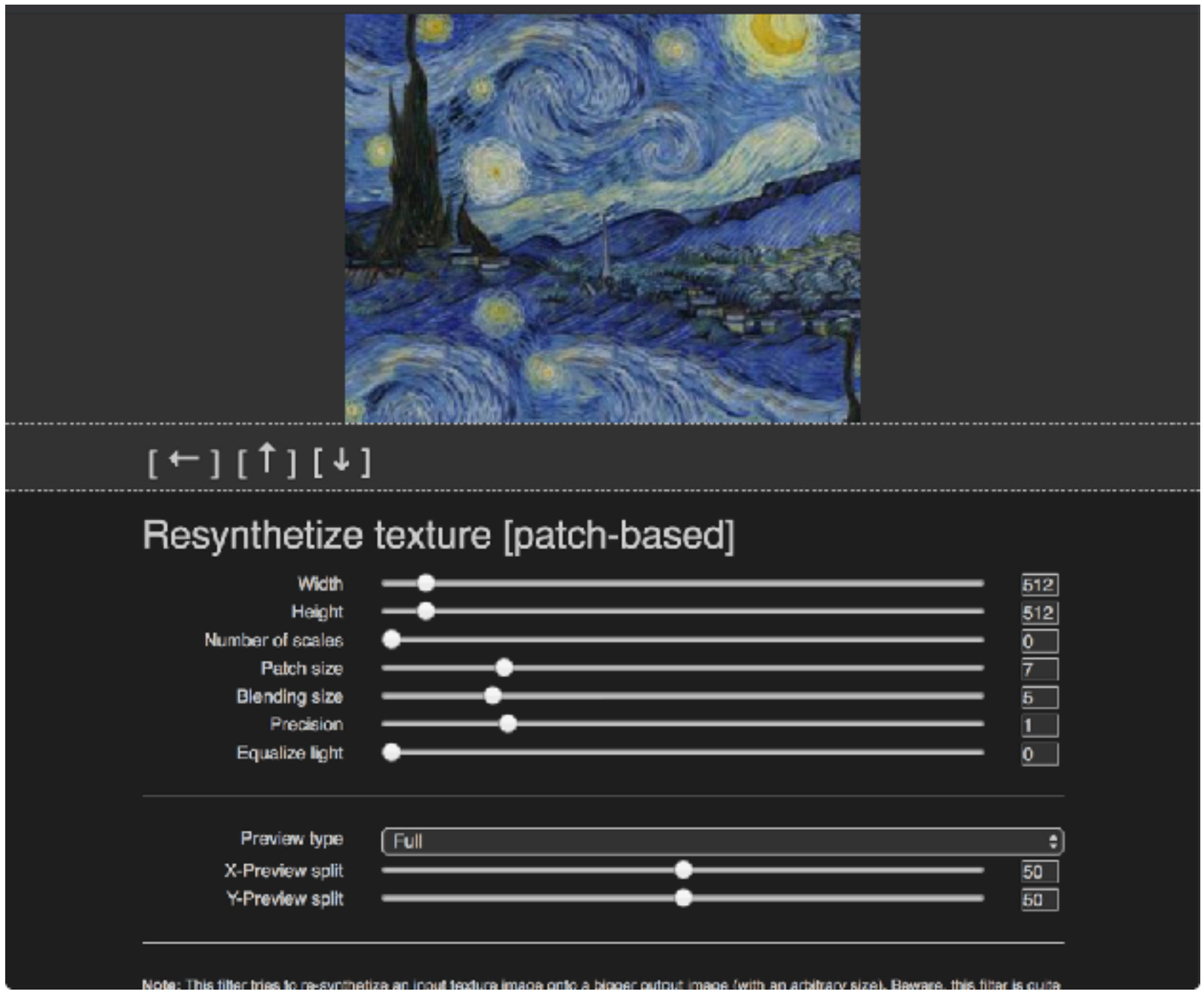
Example

Synthesis

(G'MIC implementation by D. Tschumperlé)

- **Pros:** fast, simple (but need multi-scale version of PathMatch)
- **Cons:** this is not innovative (large local copies), no statistical control, difficult to define a good initialization

Demo G'M!CoL



https://gmicol.greyc.fr

filter / Patterns / Resynthetize texture [patch-based]

Exemplar-based Texture Synthesis

PatchMatch approach (sequential synthesis of patch) has been investigated before for texture synthesis and transfer (and share similar features):

- **Sequential Pixelwise Processing:** *Texture Synthesis by Non-parametric Sampling* [Efros & Leung, ICCV'99] (IPOL demo: http://demo.ipol.im/demo/59/)

- **Sequential Patch based Processing:** *Image Quilting for Texture Synthesis & Transfer* [Efros & Freeman, SIGGRAPH'01] (IPOL demo: http://demo.ipol.im/demo/171/)

- **Global Patch based optimization:** using GraphCut [Kwatra et al. SIGGRAPH'03], and constrained Patch matching [Kwatra et al. SIGGRAPH'05]

We will only study [Kwatra *et al.*'05] that is used in State of the Art methods (« *Image Melding* » [Darabi *et al.*'12] and « *Self Tuning Texture Optimization* » [Kaspar *et al.*'15])

Exemplar-based Texture Optimization

- Patch-based **variational formulation** of « *Texture Optimization for Example-based Synthesis* » [Kwatra et al. SIGGRAPH'05]

- **Problem:** compute a synthetic image u that is a **local minimum** of the energy $E(u)$ (here patches p_0 are from example image u_0)

$$\arg\min_{u} \left\{ E(u) := \min_{f} \sum_{x \in \Omega'} \|p(x) - p_0(f(x))\|^r \right\}$$

$$\textit{where} \quad p(x) = (u_i(x+t))_{i \in \{1,\dots,d\}, t \in \omega}$$

- **Ill posed problem** as $E(u_0) = 0$ (we don't want the **optimal** solution !) and it is **non-convex** (joint minimization over f and u)

Exemplar-based Texture Optimization

Optimization: iterative **alternate** minimization (in **multi-level fashion**)

- **Patch Matching:** minimization over f (u being fixed)

$$f(x) = \arg\min_{y} \|p(x) - p_0(y)\|$$

To accelerate (and improve result), only subsampled are considered: $x \in \Omega' \subset \Omega$

Remark: *in parallel, but could eventually be accelerated by PatchMatch*

- **Patch Aggregation:** minimization over u (f being fixed)

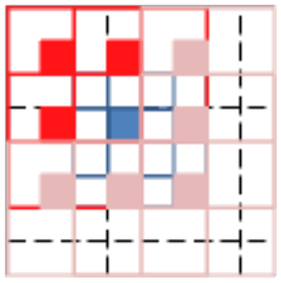
if $r{=}2$*:* simply averaging pixels from overlapping patches

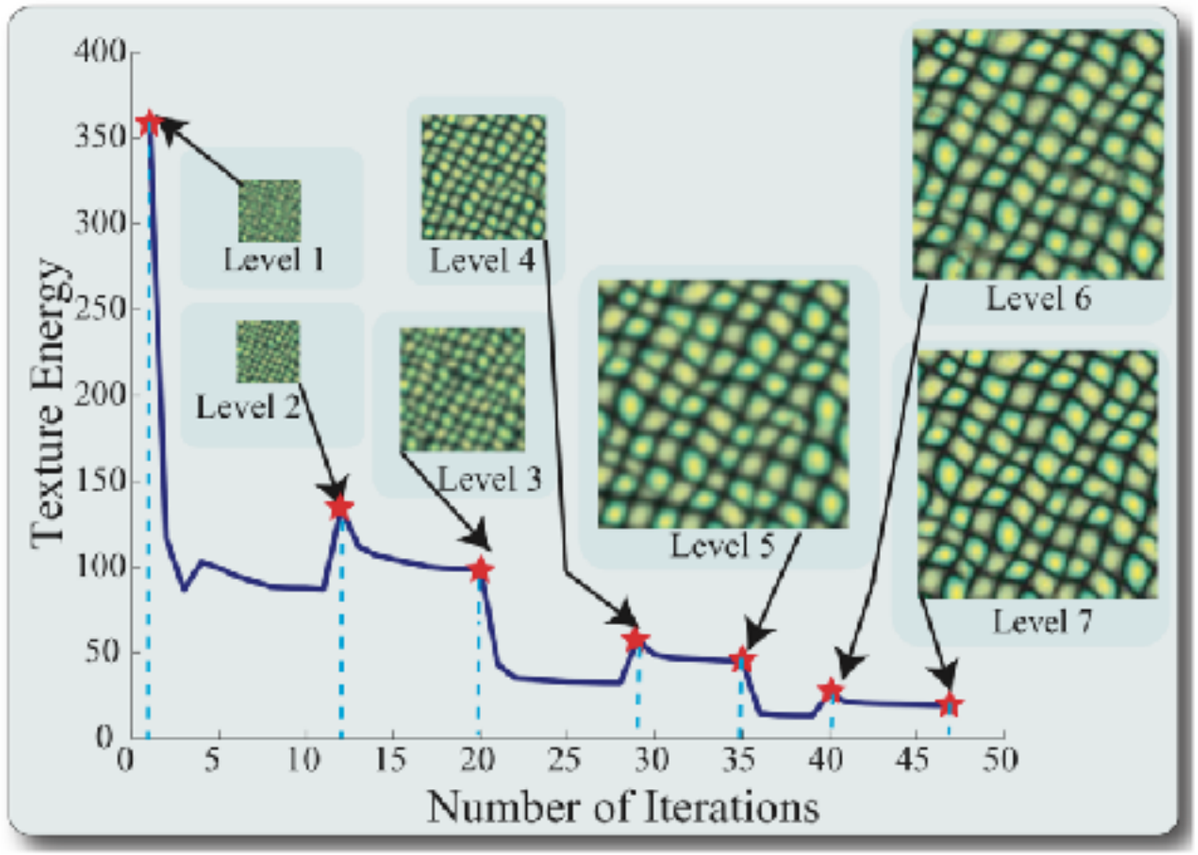
(like block NL-means or NL-PCA) $$u_i(x) = \frac{1}{|\omega|} \sum_{t \in \omega} p_{0,i}(f(x-t))|_t$$

but in practice $r{=}0.8$ is used (requiring **reweighed least square algorithm**)

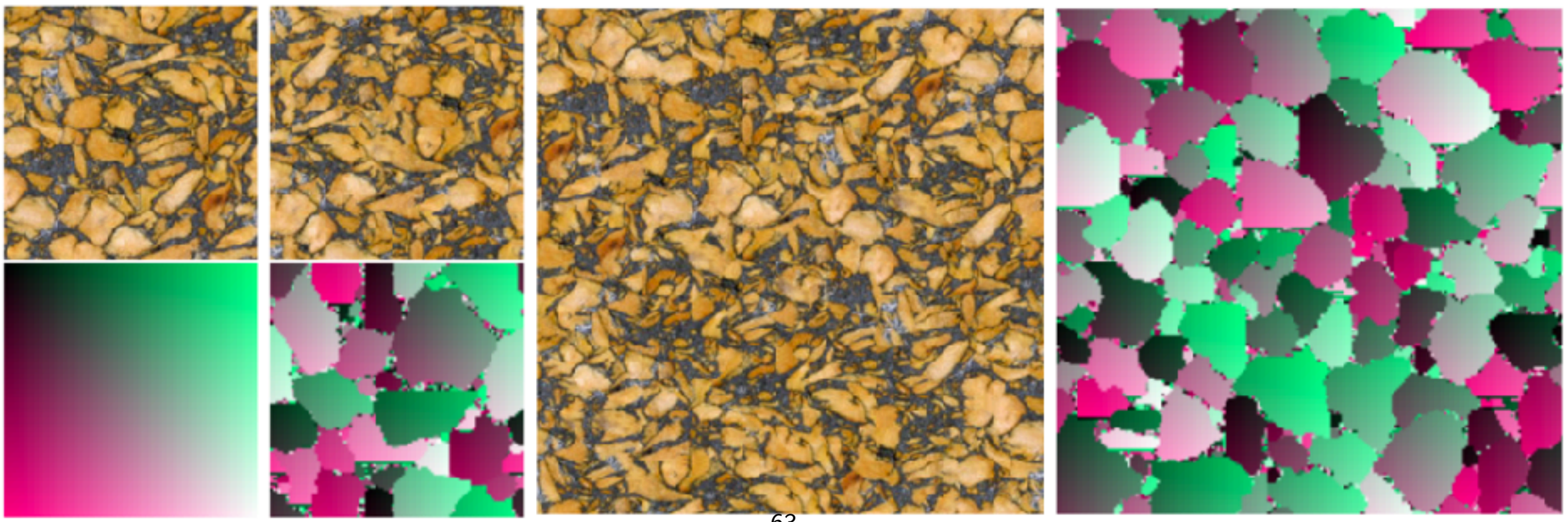
In [Gutierrez *et al.*'07], we showed that using $r{=}1$ is as good just require **median**

- **Optimization tips & tricks:**
  - **multi-level optimization** using gaussian pyramid of the example, and **multiple patch size** to capture long distance correlation
  - **Random initialization**
  - Subsampling of patch (stride)





- **Results:** piecewise offset nearest neighbor field !

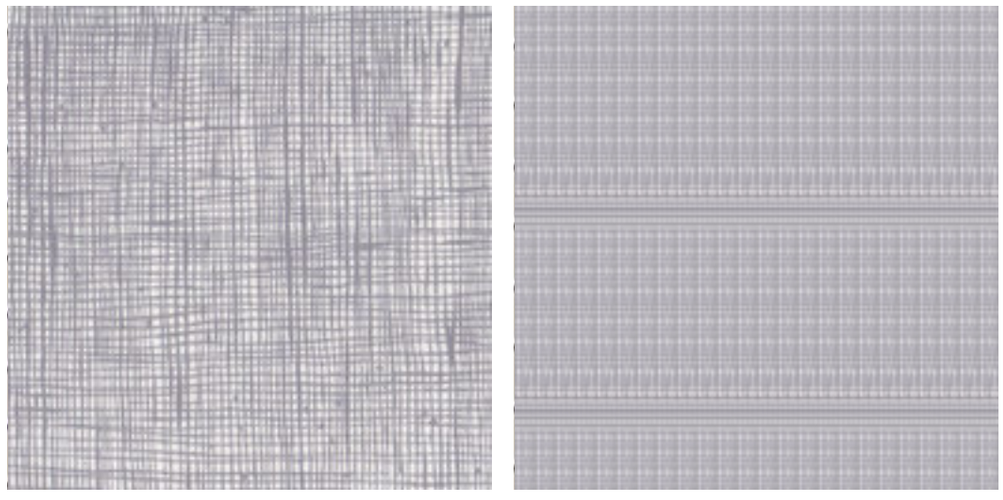


Illustrations source: Gutierrez and Kwatra

Exemplar-based Texture Optimization

- **Results**

example *synthesis*



- **Pros:** variational method where constrained can be added (e.g. [Gutierrez et al.'07]), can be accelerated with **PatchMatch** [Kaspar et al.'15]

- **Cons:** slow, and not innovative (large local copies) can « grow garbage »

Demo Notebook

https://sites.google.com/site/rabinjulien/enseignement

Texture_optimization_TP_exercice.ipynb

Adding patch assignment constraint

- **Statistical control:** We can impose [Gutierrez *et al.*'17] that every example patch is used once to avoid loosing information:

$$f(x) = \arg\min_{\sigma \in \Sigma} \|p(x) - p_0(\sigma(x))\|$$

where Σ is the set of permutation of pixels coordinates

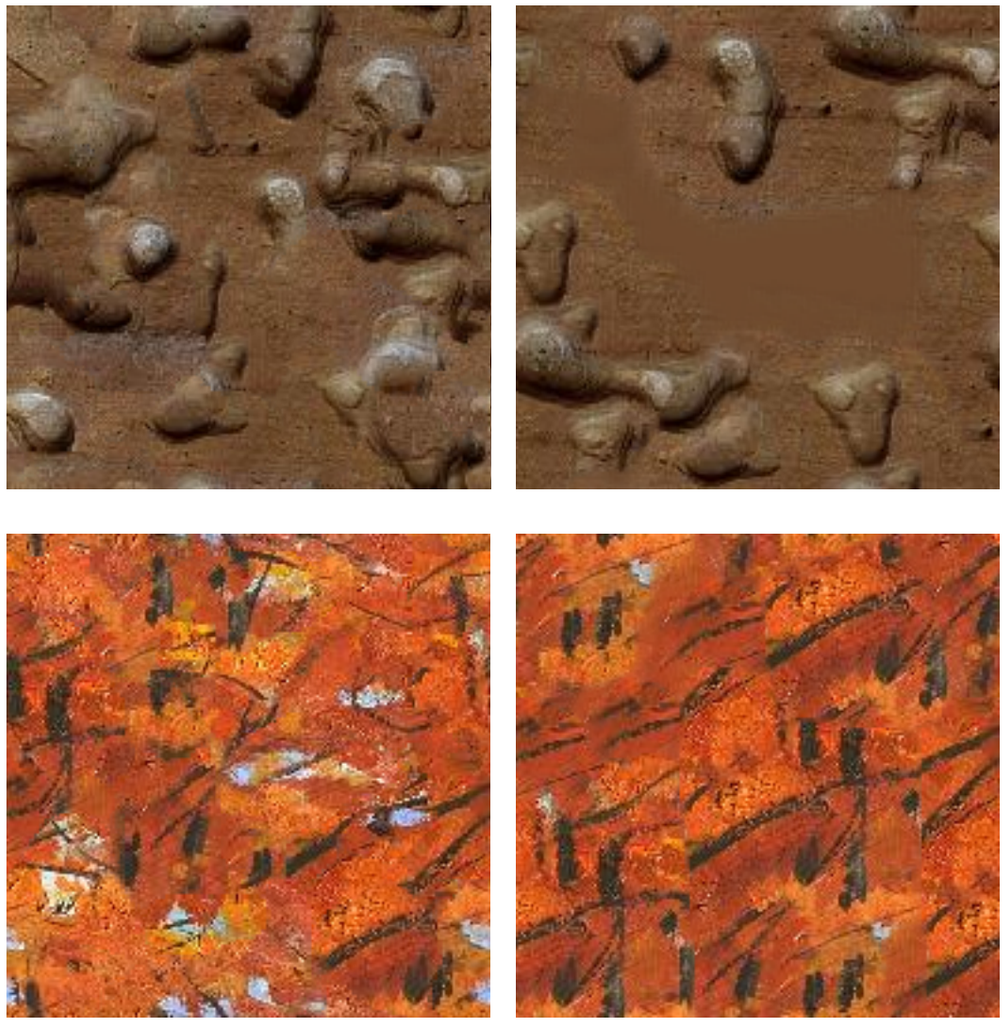
- **Optimization:** it is **not separable** anymore (non-parallel) !
  It boils down to solve a Linear Sum Assignment Problem (LSAP: see course from Luc Brun) using for instance the Hungarian algorithm (see [Bougleux'18]).
  In matrix form, we have:

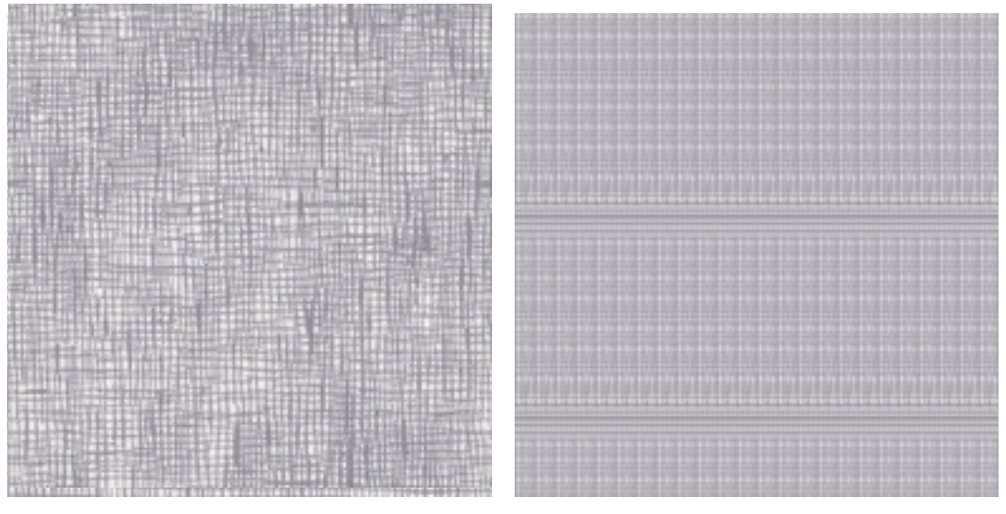
$$\arg\min_{P} \sum_{x,y} P_{x,y} \|p(x) - p_0(y)\|^2 \text{ s.t. } \sum_{x} P_{x,y} = \sum_{y} P_{x,y} = 1 \text{ and } P_{x,y} \geq 1$$

$$\text{and } \sigma(x) = y \text{ if } P_{x,y} = 1$$

Adding patch assignment constraint

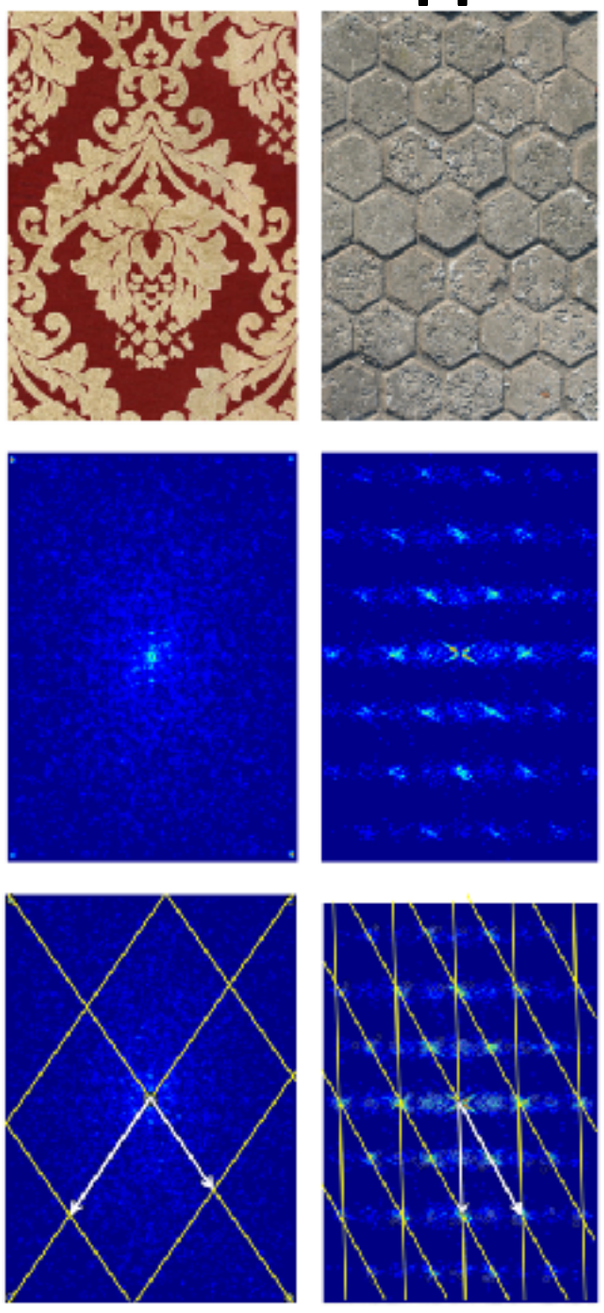
- Comparison **with** / **without** patch assignment constraint





Non stationary texture synthesis

- In « Self Tuning Texture Optimization » [Kaspar *et al.*'15], a **lattice detection approach** is used to generate a pseudo-periodic initialization

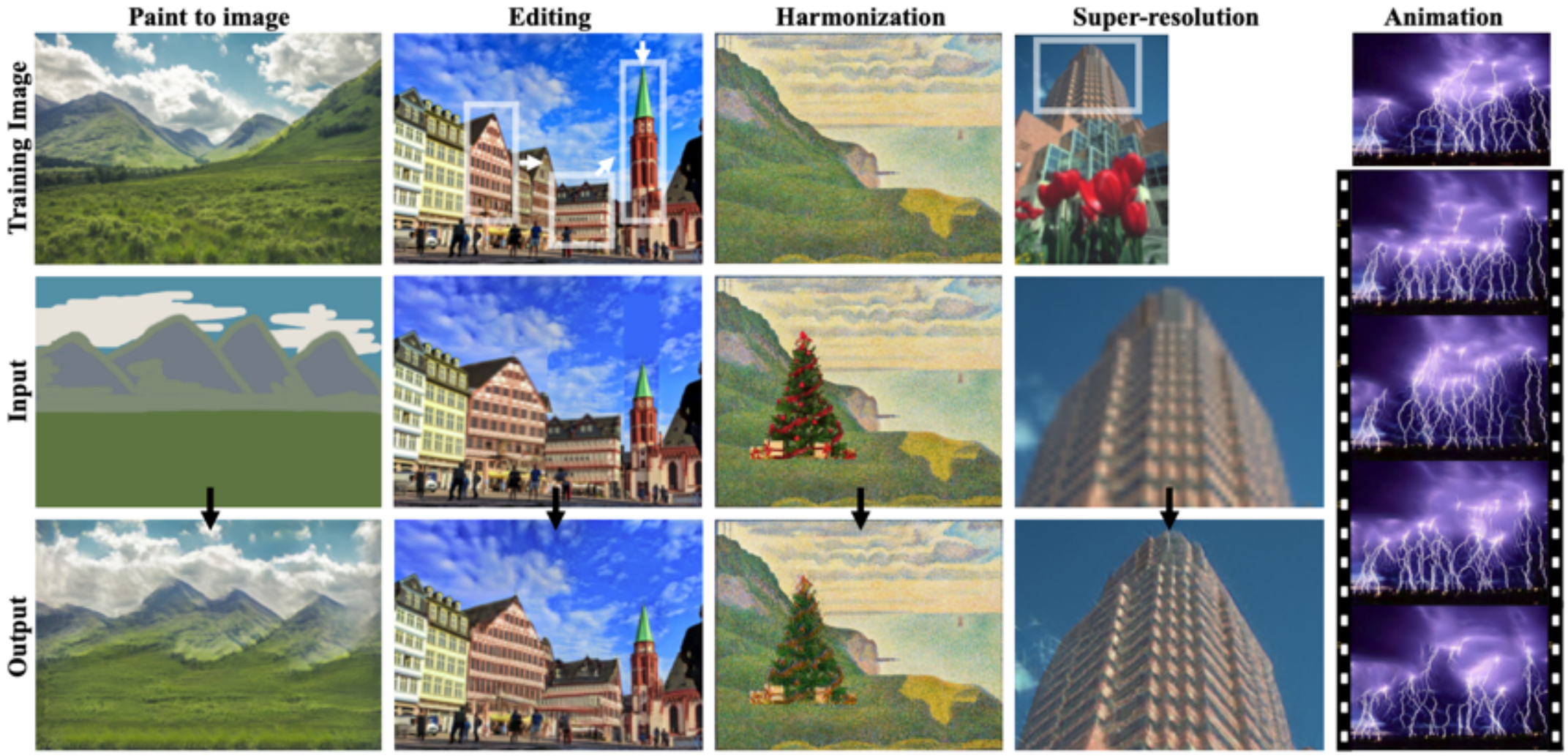


Random Initialization *Lattice Initialization*

Single-Image Generative Models

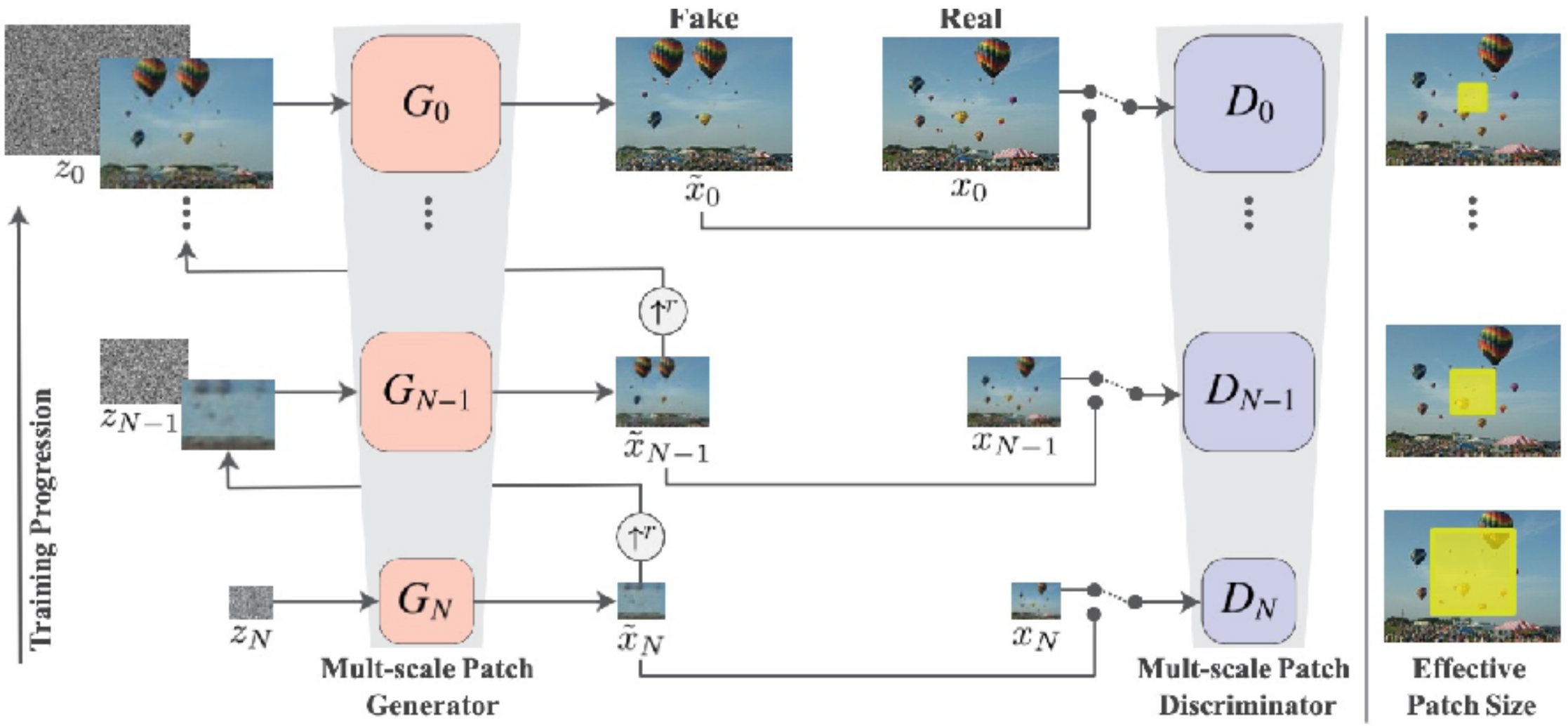
Single Image Generation

- **Goal** : like texture synthesis, sample a new realistic image from a single image example
- **No stationary hypothesis anymore !**
- **SinGAN** [Shaham *et al.*'21] :



SinGAN

- **SinGAN** [Shaham *et al.*'21] Learning a Generative Model from a Single Natural Image
- train a multi-scale **patch generator** using GANs



SinGAN

- **Optimization:** at each scale s

$$\min_{G_s} \max_{D_s} \mathcal{L}_{adv}(G_s, D_s) + \alpha \mathcal{L}_{rec}(G_s)$$

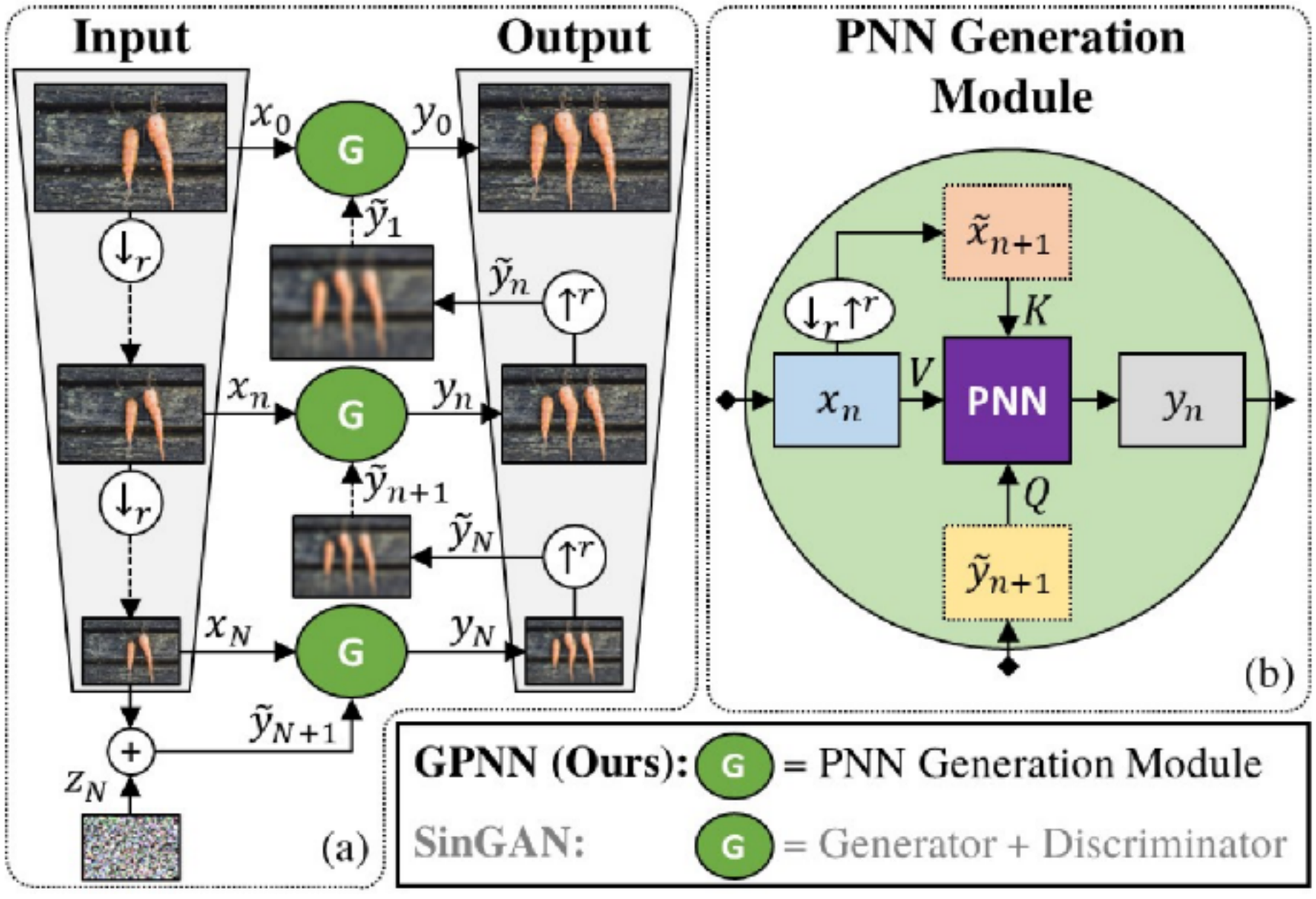
with $$\mathcal{L}_{rec}(G_s) = \|x_s - G_s(z_s | \tilde{x}_{s+1} \uparrow r)\|^2$$

- **Results** : **~1h training** on a single GTX 1080, **<1s sampling**

Examples *Random Samples*

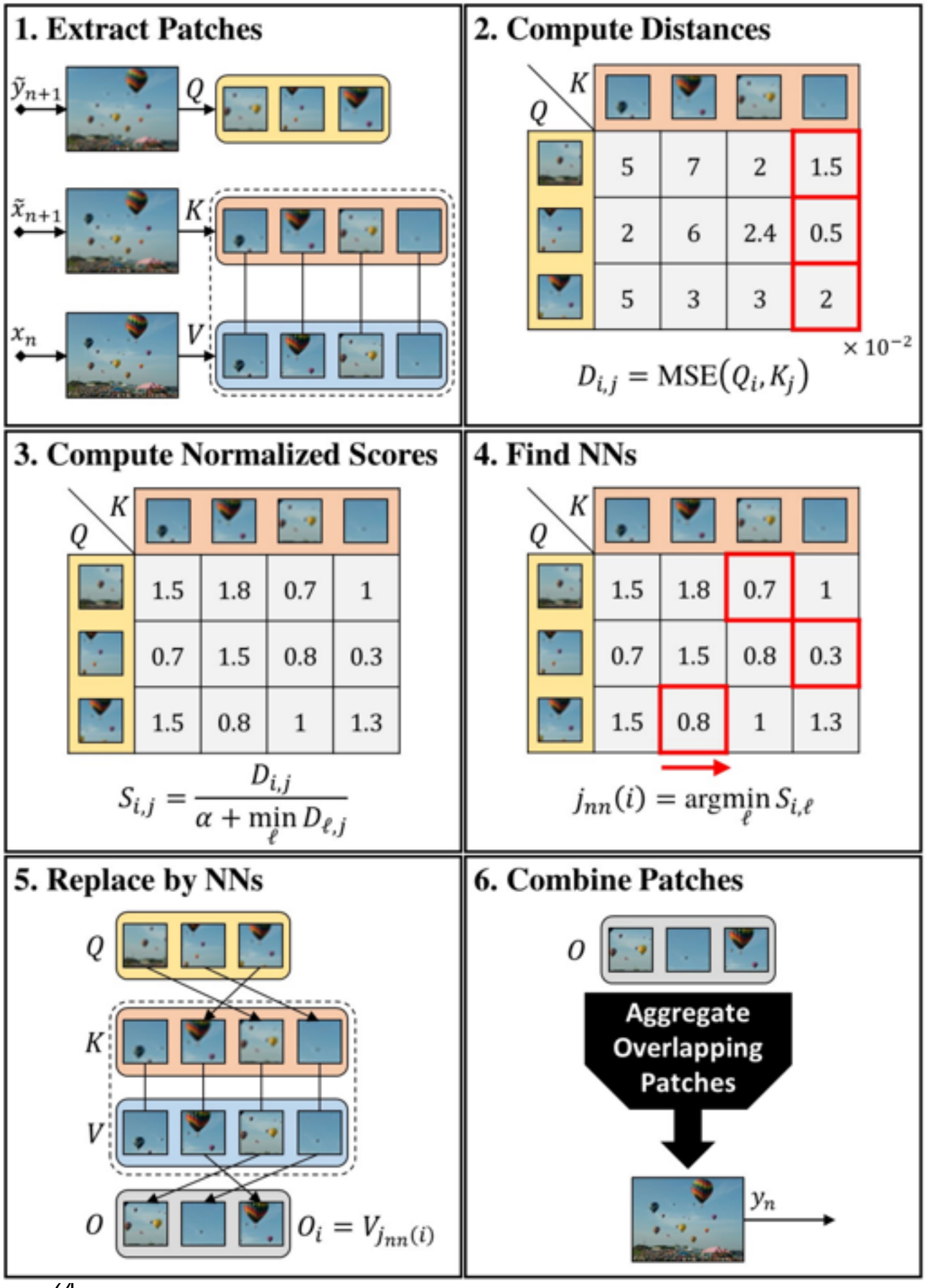
GPNN

- **GPNN:** « Drop The GAN: In Defense of Patch Nearest Neighbors as Single Image Generative Models » [Granot, et al, CVPR'22]



GPNN

- **PNN:** Patch Nearest Neighbour module inspired from (cross) attention
- **non parametric !**



Back to texture optimization with spatial constraints

- **Inspired from SinGAN:** we can constrain [Shaham *et al.*'21] that some masked pixels are closed to the original image

$$\arg\min_{u} \{E(u) + \alpha F(u, u_0)\}$$

where $\alpha > 0$ is an hyper-parameter controlling the degree of fidelity

$$E(u) := \min_{f} \sum_{x \in \Omega'} \|p(x) - p_0(f(x))\|^r$$

$$F(u) := \sum_{x \in \Omega} \|u(x) - u_0(x)\|^2$$

- **Additional (hard) constraints :** copy image on domain border $\partial\Omega$ (if images u and u_0 share the same spatial dimensions)

Back to texture optimization with spatial constraints

- **Optimization:** as before
  - first solve patch NN (patch nearest neighbour search) to compute f
  - then optimize image u by blending optimal image u for E (patch aggregation) and solution from F (u0)

$$u \leftarrow \frac{1}{1+\alpha}(u + \alpha u_0)$$

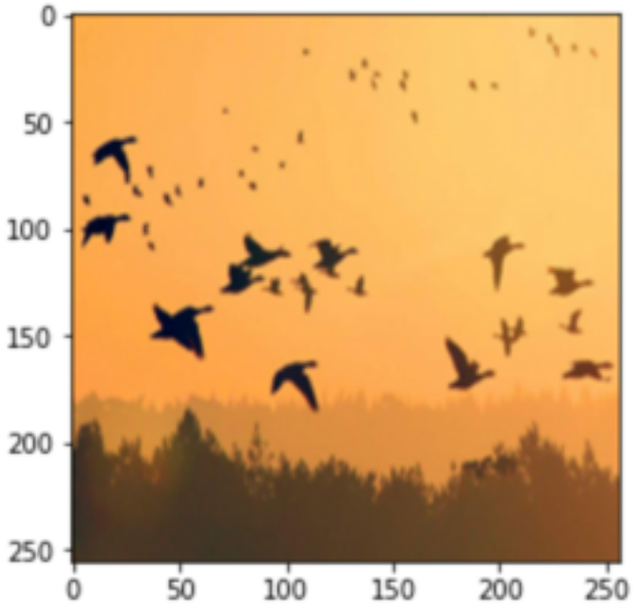
Demo Notebook

https://sites.google.com/site/rabinjulien/enseignement

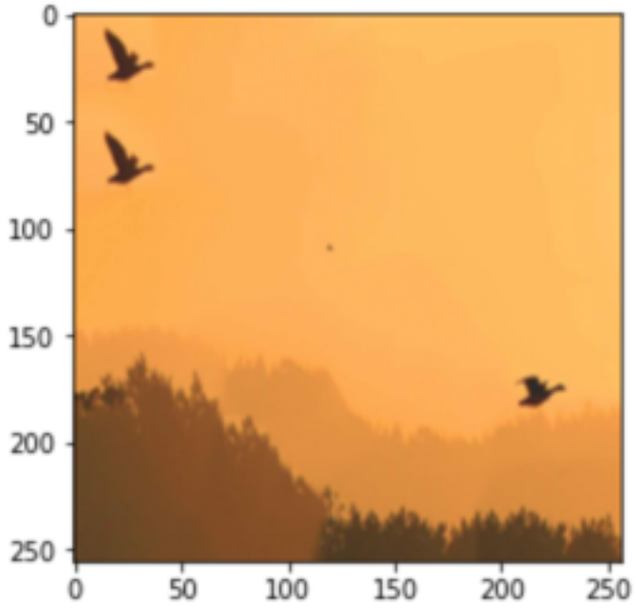
- **Demonstration :**

Texture_optimization_with_spatial_fidelity.ipynb

reference



synthesis



Style transfer

Texture transfer

- **Originally proposed in** « *Image Quilting for Texture Synthesis & Transfer* » [Efros & Freeman, SIGGRAPH'01] and « *Patchwork texture synthesis* » [Harrison'02]

- **Idea:** Additionally to a random initialization, **use an existing image to guide the patch matching** based on geometric information (rather than texture)

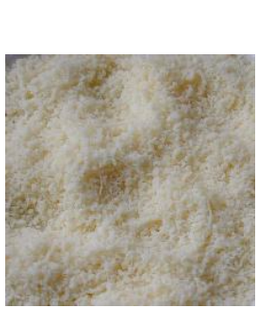
- **In practice:** Many tricks to separate texture from color and geometry information (change the metric, using gradient, contour detection, etc.)

Texture transfer

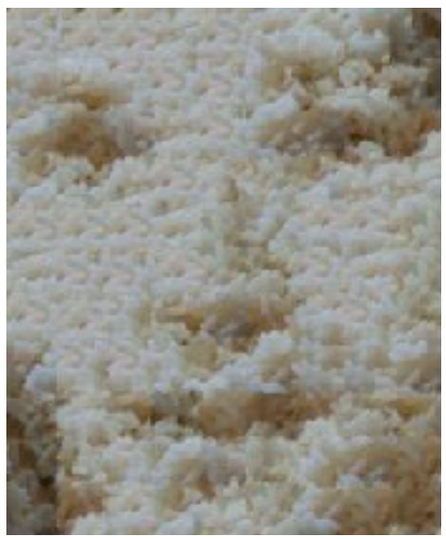
- **Examples** from [Efros & Freeman, SIGGRAPH'01]

Content

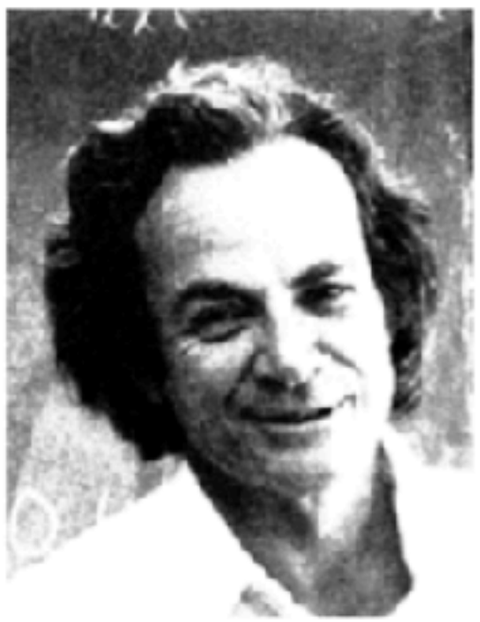
Style



Transfer



source texture



target image

correspondence maps

texture transfer result

Style transfer

- **State of the art approaches** based on patches derive either from the variational formulation or the PatchMatch approach:

  - « *Style-Transfer via Texture-Synthesis* » [Elad & Milanfar'16] extend the texture (or *style*) energy from [Kwatra *et al.*'05] to add a new *content* term comparing the generated image to the target one.

  - New G'MIC plug in from David Tschumperlé using PatchMatch strategy (Demo: https://gmic.eu/gallery/stylization.shtml)

Style Transfer with texture optimization

- **(Simplified) formulation** from [Elad & Milanfar'16]: s is the style image, c the content image

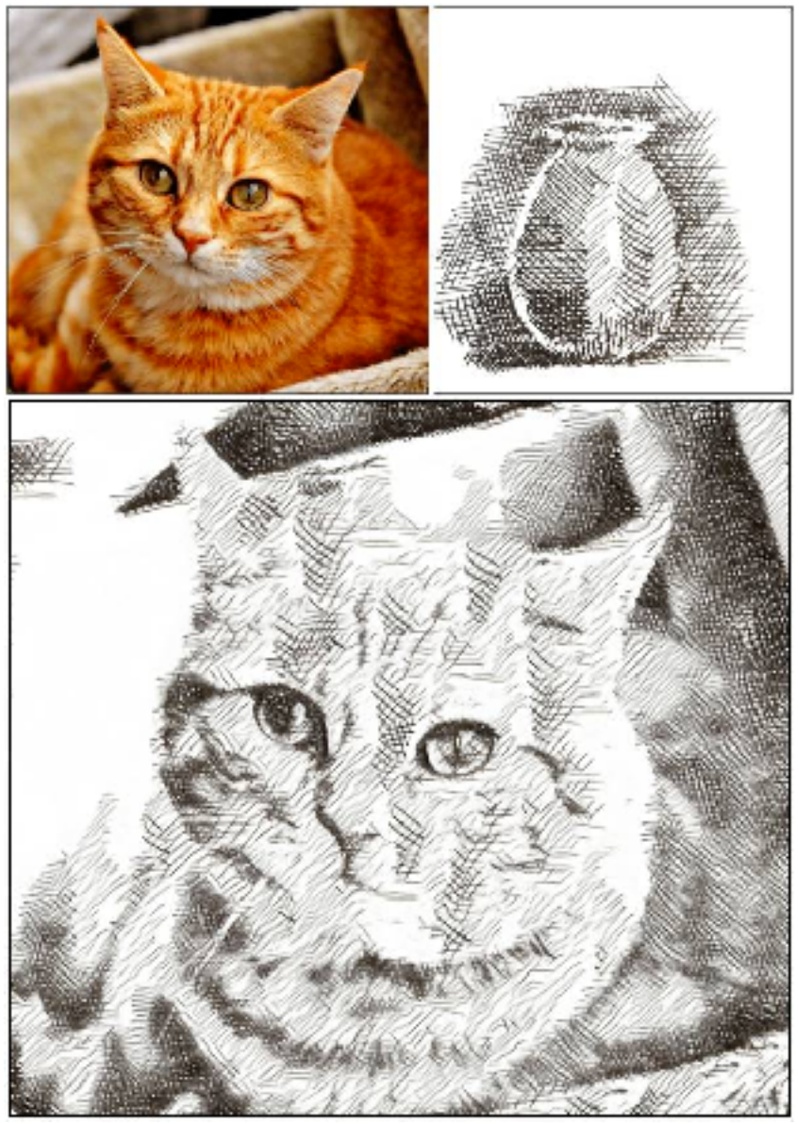
$$\arg\min_{u} \min_{f} \sum_{x \in \Omega'} \|p(x) - p_s(f(x))\|^r + \|u(x) - c(x)\|^2 + \rho(u)$$

- **Results**

Content *Style* *Transfer* *Content* *Style* *Transfer*

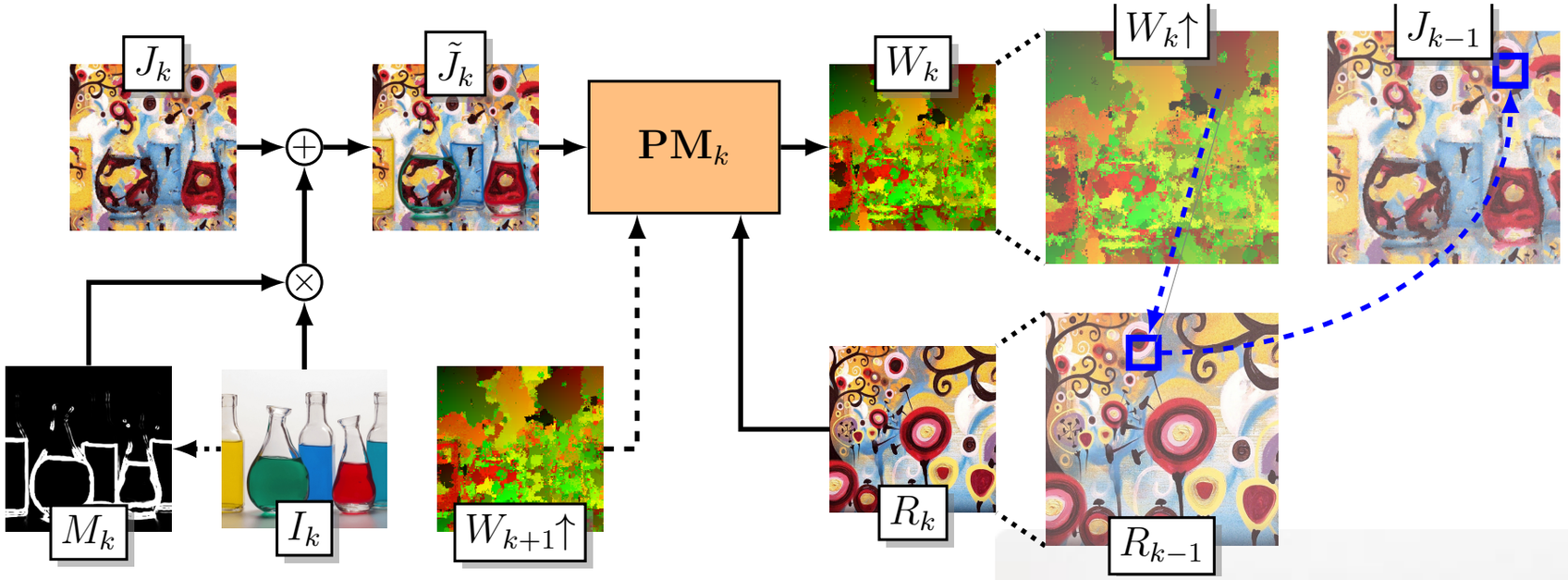
Automatic Style Transfer with patch matching

- **Results** from *Stylization algorithm* from D. Tschumperlé & B. Samuth



Automatic Style Transfer with patch matching

- **Penalised** patch-match & multi-scale pipeline



[Samuth et al', 2022]

Automatic Style Transfer with patch matching

- **Blending images** features from two images

$$\tilde{J} = (1 - M) \odot J + M \odot I$$

- *Remark* : allows to control image contour, similarly to **ControlNet** [Zang' CVPR'23] where the blending is performed with latent space by a denoising network

Input Canny edge

Default

Conclusion

- Nearest Neighbor Patch-based approaches overview:
  - **Pros:** Simple, Flexible, state-of-the-art performance (or very close to)
  - **Cons:** Still slow (cannot always benefit from GPU), fails at large scale (NN is not robust), only based on 1 exemple (what about using a large dataset ?), do not achieve **image generation**
- Next course: using machine learning for image generation