Machine Learning: from Theory to Pratice

Quality enhancement applied to videos

Benoît CHOFFIN : benoît.choffin@ensae-paristech.fr
Hamid JALALZAI : hamid.jalalzai@gmail.com
Paul ROUJANSKY : paul.roujansky@ensae-paristech.fr
Alexandre SEVIN : alexandre.sevin@ensae-paristech.fr

Friday, January 13th 2017

Attention! All our code and the datasets used are available and ready-for-demo on the following Github repository: `https://github.com/AlexandreSev/Patch-Match`

Introduction

The idea we initially wanted to dig into was to improve resolution of images during their enlargement. We noticed that this refers to two main connected principles: image denoising and - to a larger extent - image completion. We therefore decided to focus on these two topics in order to restore images to improve their quality. In a first part, we focus on image denoising through collaborative filtering/matrix factorization approaches, and in the second part on patch-based image completion.

1 Image denoising and impainting

Let us consider a noisy or degraded image X (viewed as a $n \times m$ matrix) with missing entries. The task of recovering (or perhaps "guessing" is a better term here) the missing pixels of X given its observed ones corresponds to a *matrix completion* problem. Notably, it can be seen as a specific collaborative filtering problem where we wish to fill the missing entries of X by discovering patterns within the image. This field of study has been quite recently triggered by recommendation-based problems, such as the famous Netflix prize, and a lot of new techniques have emerged since then. Let us introduce a few notations:

Let R be a $n \times m$ matrix and $\Omega \subset \{1, ..., n\} \times \{1, ..., m\}$ be the list of pairs of (i, j) such that R_{ij} is declared as "observed" for every $(i, j) \in \Omega$, i.e. Ω is the observed part of our picture. We define the projection operator on a given subset of $\{1, ..., n\} \times \{1, ..., m\}$ as:

$$\mathcal{P}_\Omega(R_{ij}) = \begin{cases} R_{ij} & \text{if } (i,j) \in \Omega \\ 0 & \text{otherwise} \end{cases}$$

Referring to our initial problem, we can therefore see R as the "true" image and $\mathcal{P}_\Omega(R) = X$ as the incomplete input image such that : $R = \mathcal{P}_\Omega(R) + \mathcal{P}_\Omega^\perp(R)$. The general goal is to recover R with the best possible accuracy, given its observed part X.

Note: we first consider here the case of a gray-scale image but the problem can be extended to colored images by applying the following algorithms to each one of the 3-depth-dimension RGB input matrices.

1.1 A first approach: SVT

Candès and Recht[CR09] proved that it was possible to exactly recover R even with a rather small number of observed entries $|\Omega|$ under the strong assumption that the rank of R matches a given constraint and is therefore not too high. In particular, let us assume that $n = m$ such that R is a $n \times n$ matrix of rank r. They showed that, under the strong assumption that there exists $C \in \mathbb{R}^+$ s.t. $|\Omega| \geq Cn^{6/5} r \log(n)$, R could be perfectly recovered by solving the following optimization problem

$$\begin{cases} \underset{Z}{\text{minimize}} & \|Z\|_* \\ \text{subject to} & \mathcal{P}_\Omega(R) = \mathcal{P}_\Omega(Z) \end{cases} \tag{1}$$

where $\|Z\|_* = \sum_{j=1}^{n \wedge m} \sigma_j(Z)$ is the nuclear norm of Z, which is the sum of its singular values. Fazel et al.[Faz02] showed that $\|Z\|_*$ is the convex envelope (i.e. the largest convex underestimator) on the spectral unit ball of rank(Z), which is a nonconvex function. Therefore, problem (1) can be seen as the tightest convex relaxation of the following problem:

$$\begin{cases} \underset{Z}{\text{minimize}} & \text{rank}(Z) \\ \text{subject to} & \mathcal{P}_\Omega(R) = \mathcal{P}_\Omega(Z) \end{cases} \tag{2}$$

Problem (2) is nonconvex and NP-hard (combinatorially hard in general for a given Ω). However, Candès et al. showed that problems (1) and (2) are equivalent under some assumptions. Cai et al.[CCS10] proposed a first-order singular-value-thresholding algorithm called `SVT` to solve (1). Notably, this algorithm is scalable to large matrices. Other algorithms that aim at solving (1) have emerged since `SVT`. The assumption biding both Ω and the rank r of Z states that for "small values of the r, e.g. when $r = \mathcal{O}(1)$ or $r = \mathcal{O}(log(n))$), one only needs to see on the order of $n^{6/5}$ entries (ignoring logarithmic factors) which is considerably smaller than n^2 the total number of entries of a squared matrix" (quoting [CCS10]).

While this solution can be really satisfactory from an intellectual point of view, it can however lead to serious over-fitting as the solution requires a null training error.

1.2 A better approach: SOFT-IMPUTE

Mazumder et al.[MHT10] proposed to reformulate problem (1) by:

$$\begin{cases} \underset{Z}{\text{minimize}} & \|Z\|_* \\ \text{subject to} & \|\mathcal{P}_\Omega(R) - \mathcal{P}_\Omega(Z)\|_F^2 \leq \delta \end{cases} \tag{3}$$

where $\|.\|_F^2$ is the Frobenius norm and $\delta \leq 0$ is a regularization parameter that controls the training error.

Instead of recovering the observed part of R exactly, as suggested earlier, the idea is rather to allow some noise in our prediction through a regularization parameter (δ) that literally controls the tolerance in the training error. The authors use the same relaxation of the rank as previously. It is worth noting that using the rank as the objective function to be minimized is rather natural in this context, as we could believe that Z lies in a much lower dimensional manifold that $n \wedge m$. Minimizing the rank hence guarantees that we do not fall in over-fitting and that the predicted image makes statistical sense.

Problem (3) can be rewritten in Lagrange form as:

$$\underset{Z}{\text{minimize}} \quad \left\{ \tfrac{1}{2} \|\mathcal{P}_\Omega(R) - \mathcal{P}_\Omega(Z)\|_F^2 + \lambda \|Z\|_* \right\} \tag{4}$$

where $\lambda > 0$ is a well-chosen regularization parameter.

The following lemma is relevant:

Lemma 1 *([Cai et al., 2010]). Let the SVD of X be $U\Sigma V^\top$. The solution of the optimization problem*

$$\underset{Y}{\textit{minimize}} \quad \left\{ \tfrac{1}{2} \|X - Y\|_F^2 + \lambda \|Y\|_* \right\}$$

is a low-rank matrix given by

$$S_\lambda(X) = U(\Sigma - \lambda I)_+ V^\top,$$

where $[(A)_+]_{ij} = \max(A_{ij}, 0)$.

$S_\lambda(.)$ is called the soft-thresholding operator.

Mazumder et al. therefore propose to solve (4) through the following proximal-gradient algorithm called `soft-Impute`:

Algorithm 1: Soft-Impute

Input: $Z_{old} = 0_{(n \times m)}$
for $\lambda_1 > ... > \lambda_K$ **do**
  break $\hookleftarrow$ `False`
  repeat
    - $Z_{new} \hookleftarrow S_{\lambda_k}\big(\mathcal{P}_\Omega(R) + \mathcal{P}_\Omega^\perp(Z_{old})\big)$
    - **if** $\|Z_{new} - Z_{old}\|_F^2 \,/\, \|Z_{old}\|_F^2 < \epsilon$ **then** break $\hookleftarrow$ `True`;
    - $Z_{old} \hookleftarrow Z_{new}$
  until *break* = ***True***;
  $\hat{Z}_{\lambda_k} \hookleftarrow Z_{new}$
end
Output: Sequence of solutions $\hat{Z}_{\lambda_1}$,...,$\hat{Z}_{\lambda_K}$

They show that the output sequence of $\hat{Z}_{\lambda_k}$ of Algorithm 1 converges to a limit Z_∞ that solves (4).

At every iteration, a warm start is given by the previous computation. As a consequence, the warm start tends to improve with the number of iterations, and therefore these tend to be gradually faster. Hastie proposes[1] to use a logarithmic decreasing sequence of regularization parameters such that λ_1 is equal to the highest singular value of $\mathcal{P}_\Omega(R)$ and λ_K is equal to 1. Doing this considerably improves the speed of convergence of Algorithm 1.

Plus, by noticing that

$$\begin{array}{rcccc} \mathcal{P}_\Omega(R) + \mathcal{P}_\Omega^\perp(Z_{old}) & = & \big(\mathcal{P}_\Omega(R) - \mathcal{P}_\Omega(Z_{old})\big) & + & Z_{old} \\ & = & \text{Sparse matrix} & + & \text{Low rank matrix} \end{array}$$

one can speed up the computation of the SVD's in the soft-threshold operation by using such implementation as the PROPACK algorithm (Larsen, 2004, 1998). As a quick note, we used `numpy.linalg.svd` that implements LAPACK routine `_gesdd` and does not use this trick.

Towards a generalization of soft-Impute:

Mazumder et al. also proposed to use alternative penalties than the nuclear norm in order to relax the initial rank constraint of the original problem:

$$\underset{Z}{\text{minimize}} \quad \|\mathcal{P}_\Omega(R) - \mathcal{P}_\Omega(Z)\|_F^2 + \lambda \text{rank}(Z) \tag{5}$$

with $\lambda > 0$ is a well-chosen regularization parameter. Indeed - as an analogy - while the l_1 norm is widely used as a surrogate for the l_0 penalty, it however tends to over-estimate the number of non-zero coefficients when the underlying model is very sparse. Instead of the nuclear norm used in (4), Mazumder et al. propose to use $\sum_j p(\sigma_j(Z); \mu)$, where p is a concave function $p(|t|; \mu)$ where μ controls the concavity of p in $|t|$. Amongst penalties, one can choose for instance the l_0 penalty, and the problem we wish to solve is then:

$$\hat{Z} = \underset{Z}{\text{argmin}} \left\{ \frac{1}{2} \|\mathcal{P}_\Omega(R) - \mathcal{P}_\Omega(Z)\|_F^2 + \lambda \sum_j \mathbb{1}(\sigma_j(Z) \neq 0) \right\} \tag{6}$$

[1] `https://web.stanford.edu/~hastie/swData/softImpute/vignette.html`

The solution of (6) - which is now nonconvex - is given by Mazumder et al. [MHT09]: it consists in a reduced-rank SVD of $\mathcal{P}_\Omega(R)$. Indeed, for every λ there is a corresponding $q = q(\lambda)$ number of singular-values to be retained in the SVD decomposition. The hard-thresholding operator resulting from (6) is:

$$S_\lambda^H(W) = UD_qV^\top \quad \text{where} \quad UDV^\top \quad \text{is the SVD of } W \text{ and } \quad D_q = \text{diag}(d_1, ..., d_q, 0, ..., 0)$$

(6) can be solved similarly as (4) through a descent scheme (`Hard-Impute`) also proposed by Mazumder et al.[MHT10]. This algorithm tends to outperform `soft-Impute` in low noise situation ; otherwise, it appears to be "too aggressive in selecting the singular vectors from the observed entries and hence ends up reaching a very sub-optimal subspace" [MHT10].

1.3 A different approach: Maximum Margin Matrix Factorization (`MMMF`)

Srebro and al. [SRJ04] proposed to consider a rank-r approximation of the solution Z given by $Z = AB^\top$ where A is $n \times r$ and B is $m \times r$ - further hypothesis can be made on the decomposed matrices A and B such as sparsity and non-negativity. If r is low, then A and B are low-ranked. They then use a very handy lemma which states that:

$$\|Z\|_* = \min_{Z=AB^\top} \|A\|_F \|B\|_F = \min_{Z=AB^\top} \frac{1}{2}(\|A\|_F^2 + \|B\|_F^2)$$

Therefore, the problem boils down to a rewriting of (4) using matrix factorization as a decomposition method of Z. The so-called maximum-margin matrix factorization (`MMMF`) optimization problem reads:

$$\underset{A,B}{\text{minimize}} \quad \tfrac{1}{2} \left\| P_\Omega(R - AB^\top) \right\|_F^2 + \tfrac{\lambda}{2}(\|A\|_F^2 + \|B\|_F^2) \tag{7}$$

It appears that this problem is not convex but bi-convex, and can be solved by using an alternating least-squares (ALS) algorithm.

A remarkable result is that the space of solutions of `MMMF` contains the solution of `soft-Impute`. Indeed, it is clear that `MMMF` searches a solution in a higher space that includes `soft-Impute`. For a sufficiently large rank r though: `MMMF` $\equiv$ `soft-Impute`. Yet, `MMMF` is much slower than `soft-Impute` in practice and the tuning of the regularization hyperparameters of the ALS is not trivial in the case of image completion.

1.4 Perhaps the best algorithm: `softImpute-ALS`

Hastie et al. [Has+15] proposed to reduce the computational bottlebeck caused by the SVD in `soft-Impute` by combining it with the underlying root idea of `MMMF` and replacing this optimization step with a simple matrix multiplication (the trick is quite tremendous here!). Indeed, Hastie proposes to replace the soft-thresholding computation step of `soft-Impute` (see Algorithm 1), by the following optimization problem:

$$\underset{A,B}{\text{minimize}} \quad \tfrac{1}{2} \left\| \hat{X} - AB^\top \right\|_F^2 + \tfrac{\lambda}{2}(\|A\|_F^2 + \|B\|_F^2) \tag{8}$$

with $\hat{X} = \mathcal{P}_\Omega(R) + \mathcal{P}_\Omega^\perp(Z_{old})$.
The resulting algorithm is called `softImpute-ALS` and is detailed page 14 of [Has+15].

Hastie summarizes the core advantages of this algortihm as below (quoting):

- "Since $\hat{X}$ is fully observed, the (ridge) regression operator is the same for each column, and so is computed just once. This reduces the computation of an update of A or B over ALS by a factor of r.
- By orthogonalizing the r-column A or B at each iteration, the regressions are simply matrix multiplies, very similar to those used in the alternating subspace algorithms for computing the SVD.
- The ridging amounts to shrinking the higher-order components more than the lower-order components, and this tends to offer a convergence advantage over the previous approach (compute the SVD, then soft-threshold).
- Just like before, these operations can make use of the sparse plus low-rank property of $\hat{X}$"

All in all, this algorithm is amongst the most optimized for matrix completion. In addition to that, it can make use of parallelization*. The resulting algorithm is called `softImpute-ALS`.

* Indeed, in the case of a large image, the latter can be divided into chunks on which the same optimization process is applied. The local predictions for each chunk are then assembled and re-orthogonalized for reiteration until convergence is reached.

1.5 Non Negative Matrix Factorization (`NMF`)

Algorithm 2: NMF

Input: $r << (n, m), Maxiter, X$
for *i=1...Maxiter* **do**
- Solve on H $W^T W H = W^T X$
- Put negative values of H to zero
- Solve on W $H H^T W^T = X H^T$
- Put negative values of W to zero

end
Output: W, H

Non Negative Matrix Factorization (`NMF`) is an approach widely used in recommandation proposed by Berry et al.[2]. The core of this approach is again to decompose the noisy image as the product of two low-rank matrices. If we consider X a $n \times m$ matrix containing only positive values, `NMF` provides two matrices which only contain positive or null values, W of size $n \times r$ and H a $r \times m$ matrix such that $X \simeq W H^\top$. By practice, we clearly see that by selecting $r \ll min(n, m)$ we tend to have more parsimonious representations. The problem to be solved hence becomes :

$$\underset{W,H>0}{\text{minimize}} \quad \tfrac{1}{2} \left\| X - W H^\top \right\|_F^2 + \lambda \text{pen}(H, W) \tag{9}$$

with the quite abusive notation W > 0 meaning that all $W_{i,j} > 0$ with $(i, j) \in [\![1, n]\!] \times [\![1, r]\!]$ and pen a given penalization on W and H such as Frobenius norm. Berry and al. also describe that an equivalent `ALS` is smoother and tends to reach global minimum rather than local ones.

1.6 Application

1.6.1 Image denoising

We now test `soft-Impute` algorithm on different uniform salt-and-pepper toy examples.

First we consider a trivial example of a checkboard where the rank is relatively low.

[2] `http://www.public.asu.edu/~jye02/CLASSES/Fall-2007/NOTES/aNMF-rev-06.pdf`

Figure 1: Application of Soft Impute on a 20% observed gray-scale 200×200 checkboard

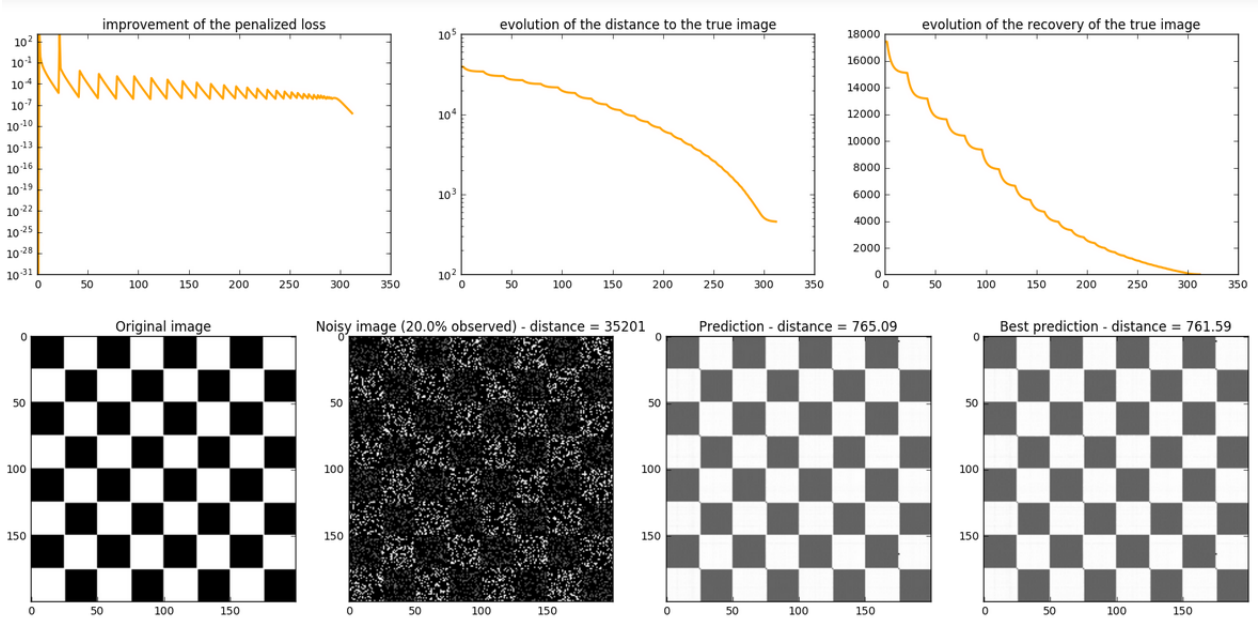
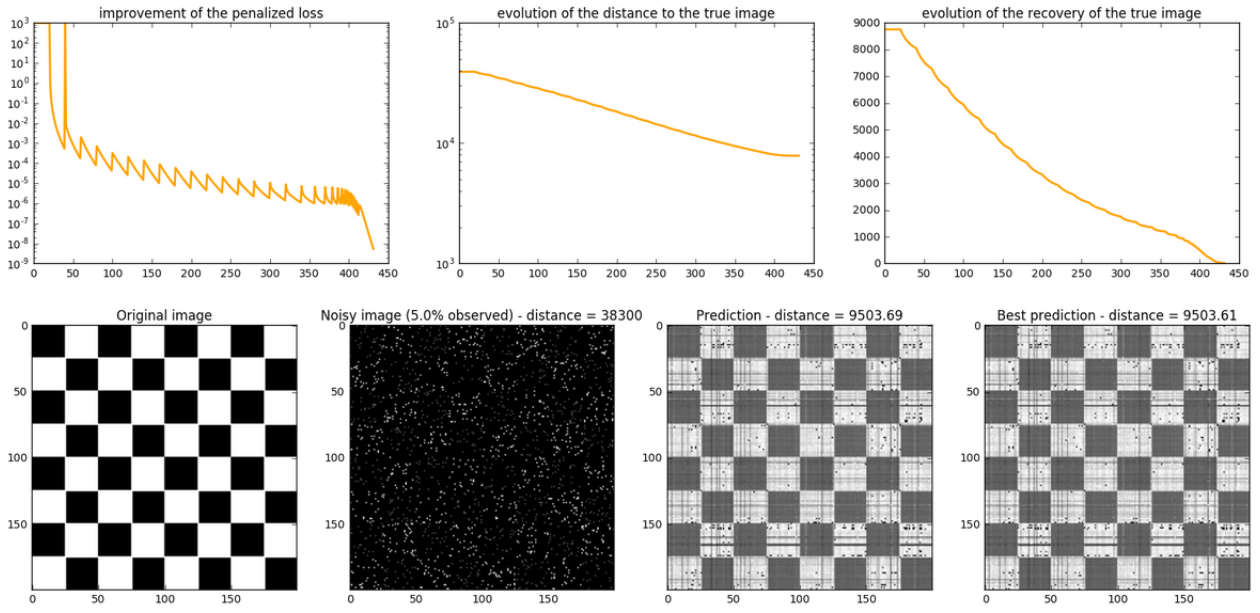


Figure 2: Application of Soft Impute on a 5% observed gray-scale 200×200 checkboard



We see that when $|\Omega|$ is too low, the recovery is not manageable anymore (when it is equal to 20%, we manage a near-perfect recovery while when it is equal to 5%, the recovery is rather bad as the rank is not tractable anymore).

We now test the algo on some toy images (respectively a black-and-white and a colored RGB image) on which we applied a uniform salt-and-pepper noise. We noise 30% of the image in each case.

We first test algorithm `soft-Impute`. The "Best prediction" is obtained by replacing the entries of the prediction by the observed entries of the input image (the difference is quite marginal).

Figure 3: Application of Soft Impute on a 30% observed gray-scale 512×512 image

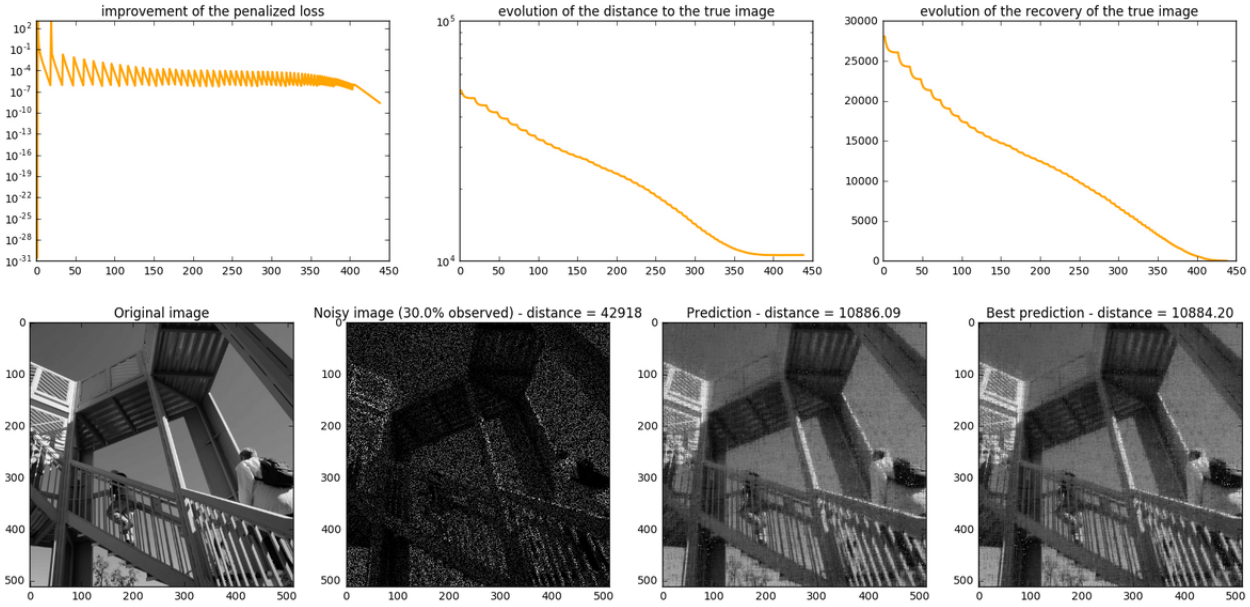
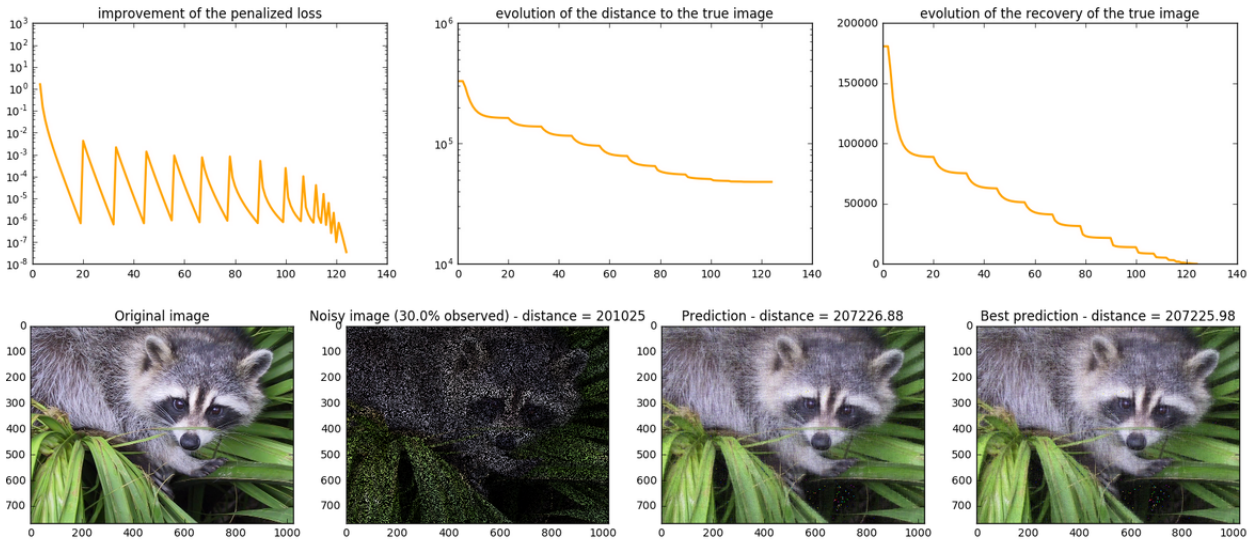
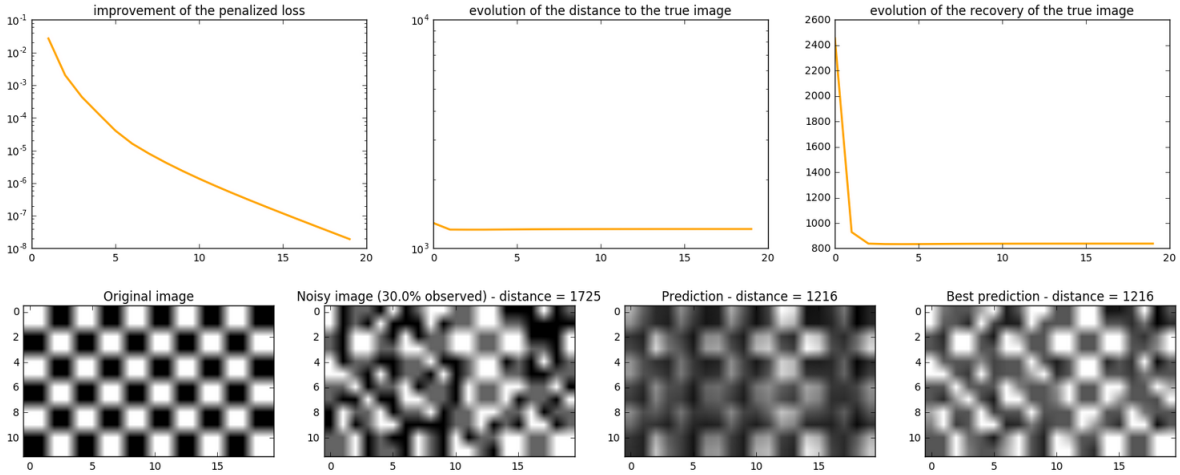


Figure 4: Application of Soft Impute on a 30% observed RGB 768×1024 image



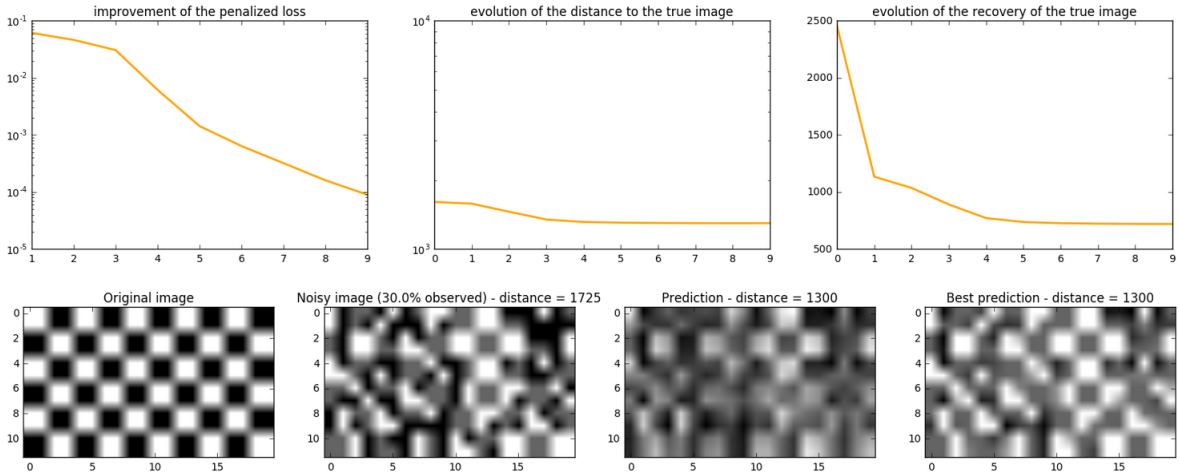
Let's now test algorithm `NMF`. First we consider the checkboard where the rank is 2 with 10 iterations.

Figure 5: Application of the NMF on a 30% observed gray-scale 12×20 image



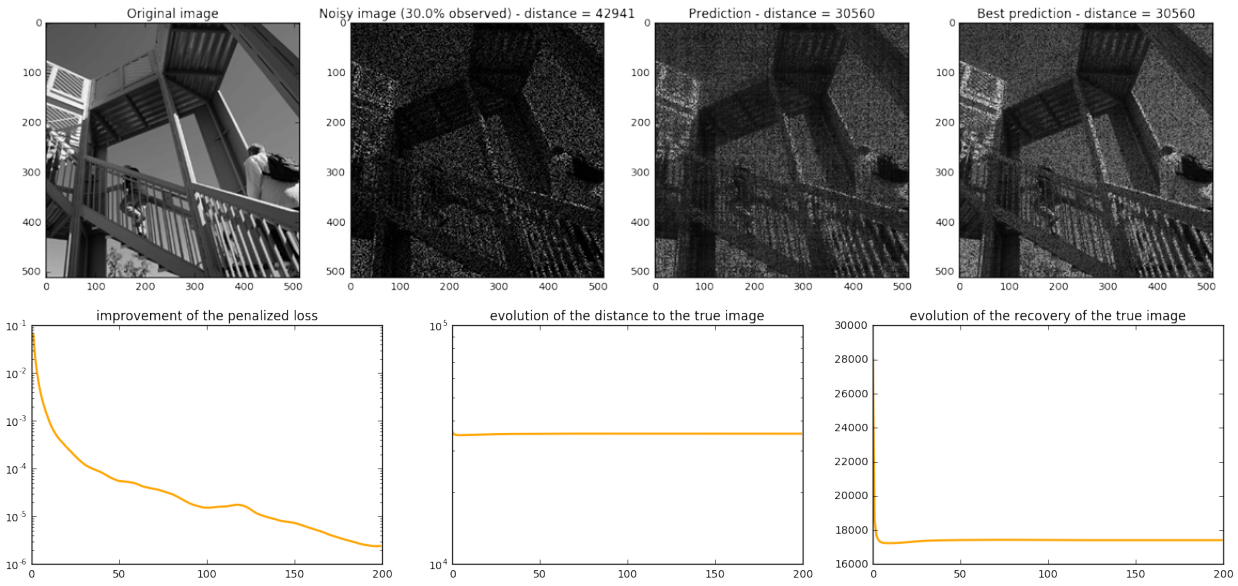
We then complete the same task considering now a rank 3 with the same number of iterations.

Figure 6: Application of the NMF on a 30% observed gray-scale 12×20 image



One can see that we don't obtain the same result though the output images are fairly similar hence the rank is an important parameter which is not always easy to guess beforehand. This is particularly true when considering videos as one can see on the following section.

Figure 7: Application of the NMF on a 30% observed gray-scale 512×512 image



1.6.2 Extension to videos

Noticing that the NMF code worked accurately enough with few iterations. It was decided to apply it to real-time video. The following process describes the steps done in our approach. The video was recorded through a regular webcam with a framerate of 10 images per second. Hence, the video appears jerky to the human eye. The clear input image from the webcam was noised up to 50% and the obtained video was treated with an NMF, with a specified rank of 20 and 10 iterations, on each of the images of the video.

The video obtained still remains highly noisy, though motion, object and human detection is highly possible. There is indeed a trade-off between the fluidity of the video and its quality. Moreover, since images are moving, the rank to process NMF on one image might no be the one suited for another image from the video. To lower computation time, we work with a set rank. It was decided to show our approach to Dr. Naty Sidaty, who advised us to do further image and treatment analysis to increase the quality of the output video for the human user. Hence, by averaging the images within the video by pairs, we managed to have a results that suits the eye even though the metrics we used showed that by doing so the image is more distant than the original input.

1.6.3 Conclusion

These collaborative filtering algorithms tend to give tremendous results in the case of image denoising in practice. Though, these algorithms tend to be computationally costly and prove to be quite inefficient when an entire region of the image is missing (compared to our initial "denoising" problem). The underlying pitfall is that these algorithms tend to produce statistically viable solutions but do not make use of "structures" within the image, which is quite essential after all. The below algorithms tend to give better results in such cases.

2 Filling missing parts within Images

Thinking about quality problems with picture, the problem of noise comes first. But if one consider old videos for example, the problem of holes in picture shows up too. The following process aims at filling holes in pictures by extracting pixels from other parts within the picture.

2.1 Exemplar-Based Image Inpainting

To tackle the problem of holes in picture, we will use Exemplar-Based Image Inpainting. This algorithm can help to get back the original picture, but it can also be used in order to remove an object from a picture. In fact, if we delete a person from a picture, this algorithm will not "find back" this person, but it will fill up the hole with a realistic background.

This algorithm is composed of three parts. First, we have to order the zone to fill. Then, we have to find a corresponding zone within the picture used to fill the chosen area. Finally, we have to mix the two areas in a coherent way. But first, we have to fix a patch size, i.e. the size of the area we will fill with one iteration. In a first approach, we fixed it, but we saw better results when we added a few randomness in this process. Therefore, we decided to choose this parameter with a random variable uniformly distributed between two bounds chosen by the user. In the following, we will call the area we are working on a patch.

Let's take a closer look at this first step. The obvious idea is to first fill the area where we have more information. It means that, if we look at each patch inside the hole or on its border, we will work on the area with the less missing pixel. This idea is relevant, but it is not enough, like it is shown in [CPT04] with the example of figure 8. Indeed, straight lines are not straight anymore. To handle this problem, the idea proposed is to extend lines first. To do so, they look at the color gradient on each pixel of the border. If the gradient is orthogonal to the border, it suggests that there is a line we have to deal with in this area.

Figure 8: Problem of using only confidence term

Source: [CPT04]

In order to use this two ideas at the same time, two coefficients are defined in [CPT04]. The first one, C, represents the confidence level of filling a patch. The second one, D, is here to first extend lines. They are defined as follows:

$$C(p) = \frac{\sum_{q \in \Psi_p \cap (\mathcal{I} - \Omega)} C(q)}{|\Psi_p|}, \qquad D(p) = \frac{|\nabla I_p^{\perp} \cdot n_p|}{\alpha}$$

where p is the pixel we are working on, Ψ_p the set of pixels in the patch of center p, $\mathcal{I}$ is the whole picture, Ω is the observed part of the picture, I_p is the maximal isophote (i.e. color gradient) of the patch, and n_p is the vector orthogonal of the border of the hole.

By multiplying these two coefficients, we get a priority coefficient. The higher this coefficient is, the sooner it will be taken into account. In fact, we will work first on the patch with the highest priority coefficient.

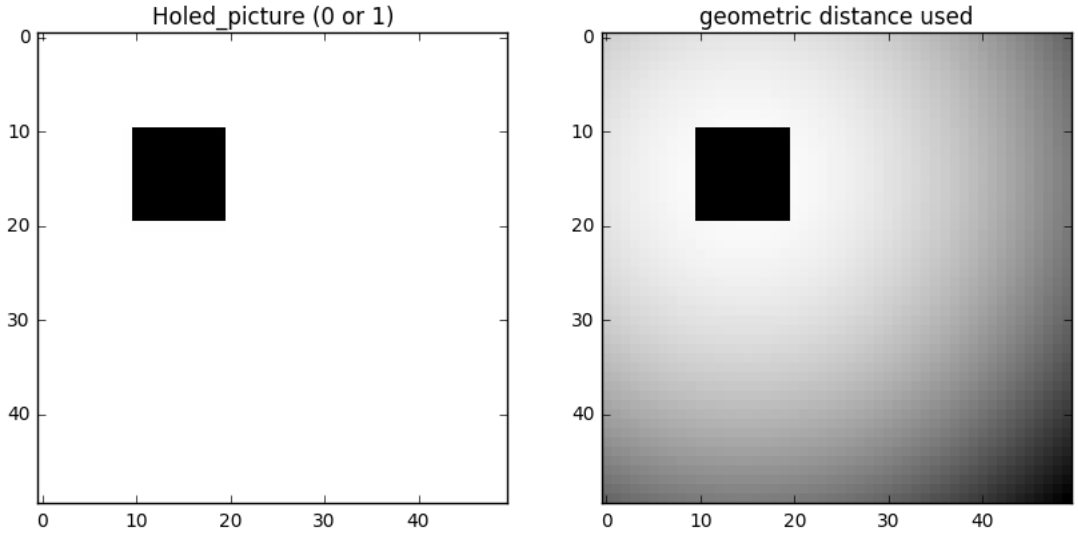
2.2 Search of the best match

At the end of the first step, we have a patch with a hole in it. During this second step, we have to find a part of the picture which matches the patch in order to use it to fill the hole. To do so, we will use a greedy algorithm. We will compare each single area with the correct size with the patch we are working on. The computational cost of this search is huge, so we decided to constraint the search area to only the neighborhood of the patch. Moreover, we used the multi-threading package of Python to use several cores.

To find the "best" match, we must define a distance. We first used the norm 2 of the difference of the two patches, computed on the observed part only of the initial patch. However, we quickly found a better way to compare two patches. In fact, when we are considering a big patch, the pixel close to the hole are more important. To represent it, we create a kind of gaussian kernel. For each pixel in the hole, we created a matrix of geometric distance of all pixels in the patch to this particular one, we transformed it to have 1 close to this pixel and 0 far away, and we averaged all the matrices obtained by all pixels in the hole. An example is shown in figure 9. Once we have this matrix, we multiply it by the difference of the two patches we want to compare, and we take the norm 2.

Finally, we compute the distance for all patches close to the patch we are working on, and we take the patch corresponding to the smallest distance. We now have two patches, a patch unfully filled and a patch considered as closed to the first one. Our goal is to mix them in order to fill the hole with the most realistic render.

Figure 9: Geometric distance used in the computation



2.3 Mixing two patches with a MinCut algorithm

Let's call the unfully filled patch the working patch and the second one, the matching patch. Our goal is to mix these two patches to fully fill the working patch. Naively, we could replace the working patch with the matching one. We could also fill only unobserved pixels from the working patch with pixels from the matching one. But these two solutions are not clever.

A successful way to deal with this issue is detailed in [Kwa+03]. To sum it up, the authors propose to consider the mixing problem as a MinCut problem in a graph context and to copy irregularly shaped patches from the input so as to apply them to the output. They apply it to texture synthesis, but we can easily adapt it to hole filing. One major advantage of this technique is that they find the optimal (and often irregular) size for the patch to be used.

At first, we need to find the optimal candidate patch from the input image. This part was detailed before in 2.2. Then, we find the optimal patch in this candidate by using a graph cut algorithm.

This texture synthesis technique is close to the one proposed by [EF01], the so-called "image quilting": small blocks from the input image are copied to the ouput image and each block is placed so as to only partly overlap with the previous ones. But we need to find the best boundary between the old patches and the new that overlap, in order to make the seam between them almost invisible. We can reformulate this problem as a MinCut in a graph, where the nodes are given by the pixels that overlap, and the vertices give a quality measure for pixels from the old and new patch:

$$M(s,t,A,B) = ||A(s) - B(s)|| + ||A(t) - B(t)||$$

where s and t are two adjacent pixels, and $A(s)$ and $B(s)$ the pixel color of s in the old patch, and in the new patch (respectively). $||.||$ denotes an appropriate norm.

We add constraint arcs, which indicate pixels linked directly to the part of their associated patch that does not overlap and give them infinite weights. These are the pixels that we absolutely want to keep from each of the patches (old and new).

However, we need to adapt this general framework to our specific problem: the figure 10 represents how we proceeded. We first apply our candidate patch to the picture with the hole, with a little amount of pixels covering the hole, while the other part of the patch overlaps with the original picture (these pixels are represented by continuous blue squares in 10). We give infinite weights to the vertices between the pixels at the border of the patch and the adjacent pixels from the original picture. The cost function giving the weight of a vertice is the same as before.

Once this has been explained, the search for the optimal seam amounts to finding the path that minimizes the cost function going from one end to another of the overlap zone, i.e. finding the minimal cut that separates the two subgraphs (the patch and the original picture).

However, this solution is not the optimal one. Indeed, what would happen if one would place a patch where another one has already been placed? It is highly probable that such a situation would create a visible seam along

the border between the old patches and the original picture. To address this issue, we follow Kwatra et al. and create a seam node between each pair of nodes where an old seam is present. Each of these new nodes is then linked to the new patch we want to incorporate and we assign these new arcs the old matching cost when this seam was created: $M(s, t, A_s, A_t)$ where s and t denote the two pixels that straddle the seam. A_s denotes the particular patch that pixel s copies from, since the overlap zone now consists in a collection of patches.

Figure 11 represents the MinCut problem when we take into account the old seams. We can easily see the old seam in green, and the seam nodes s_1 to s_5. Like we saw before, the weight of the vertice between the original image and the seam nodes (also in green) is the old matching cost, while the vertice between a seam node and its neighbor pixel is assigned the matching cost when only this pixel node is assigned the new patch. The new cut is red. If the algorithm chooses to cut a vertice between s_i and the original picture, this means that the old seam remains. In the other case, this means that the old seam (at this place) has been overwritten by the new patch. Finally, if a vertice between a seam node and a pixel node is cut, this means that a new seam (with a new seam cost) is added at the same place.

In our figure, the seam at s_1 has disappeared and the seam at s_2 has been replaced by a new seam. Furthermore, new seams have been added between 1 and 4, 4 and 5, and 6 and 9.

All of this relies on the hypothesis that finding the minimum cost path from one end of the overlap zone to the other end is equivalent to solving the MinCut problem for the graph that we built. This holds if and only if there is at most one of the three vertices originating from a seam node that is included in the MinCut; if no vertices is cut, we remove the seam. This last condition is equivalent to imposing that M verifies the triangle inequality and as consequence, the norm in M also satisfies this inequality. If this is true, picking two vertices from a seam node in the MinCut will always be costlier than picking just one of them; therefore, at most one arc is included in the MinCut. Once this condition has been checked, solving our new problem boils down to solving the same MinCut problem as before.

2.4 Avoiding brutal cuts with blurring

After several experiments, it appeared that most of the time, the recomposed image contained squared shapes where the patches were applied. In order to avoid this issue, we developed a specific procedure: at the end of each patch application, we replace each pixel pair at the border (each time, one pixel from the patch before and one pixel from the original image) by a convex combination of the two pixels. More precisely, we replace a pixel p by $\frac{2}{3} \times p + \frac{1}{3} \times p_n$ where p_n denotes its neighbor (located at the other side of the border). This procedure significantly improved our final results.

Figure 10: Schematic representation of the MinCut algorithm applied to the hole filling problem

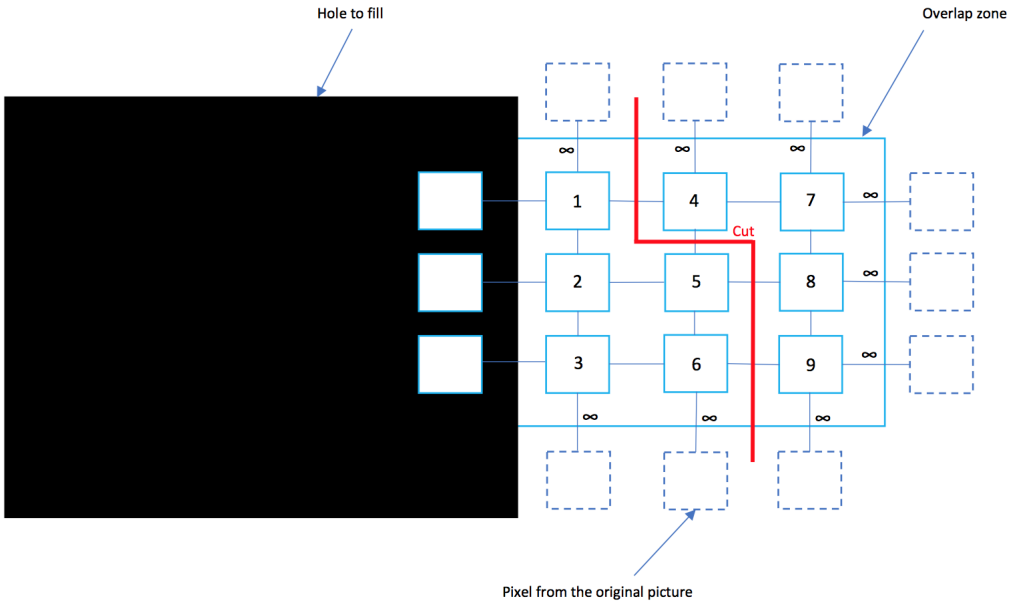
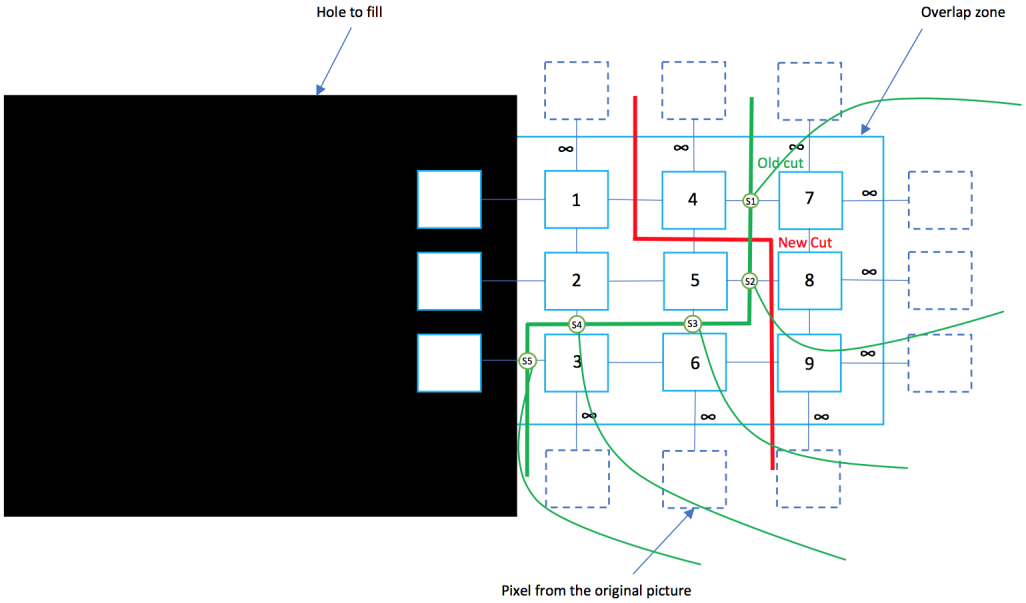


Figure 11: Schematic representation of the MinCut algorithm with old seams



2.5 Results

Once the algorithm has been implemented, we began to test it on toy pictures. We took eight pictures to test it, from the very simple line to the more complex circle or cross. We created a hole in all of this pictures, and we ran our algorithms. This lead to the figure 12.

With these tests, we notice and add a few features. To improve the results on circles, we add rotation process in the second step: we didn't only look for our patch in the rest of the picture, but also in the rest of the picture with a rotation of 90°, 180° and 270°. Thanks to this, we rebuilt almost perfectly the circles, like it is shown in figure 12.

Now that we ran these tests, we can try with some real pictures. We chose three examples, created a hole in them and ran our algorithm. The results are presented in figures 13, 14 and 15. With these examples, we can make a few remarks. First, we can see that the overall results are quite good. However, the algorithm made some mistakes. First, we can see some of the seams, which means that our blur is not powerful enough. Then, we can see on the picture of Jerusalem that the algorithm chose patches of the wall, instead of patches from the city as we could have hoped. To fix this issue, we could use an other distance to find the best match.

We could also notice that an error of our algorithm can be propagated. In fact, we forgot to remove one black pixel on the picture of the skier, and our algorithm built a small tree on it, and then a lot of small trees in the snow.

Conclusion

During this project, we tackled two different topics, both concerning image quality enhancement. The first one dealt with denoising images using matrix completion. The main idea of processes we used was to decompose the image into the product of matrices which show particular properties. Thus working on the image meant working on the matrices using an alternative approach. On a second approach, we chose to address the issue of hole completion in pictures with an algorithm of Exemplar-based Image Inpainting. Further refinements were applied, with the use of a MinCut algorithm we saw in class, to deal with visible seams along the border of the patch. We also proposed some improvement by blurring the seams at each patch application.

We implemented both methods on Python, and obtained quite good results with both techniques on real pictures. However, we did not tackle the video problem with the second algorithm, mainly because of computational issues. To improve it, we could parallelize the algorithm. We could also adapt the patch size to the picture, or also take a more global point of view to better understand the underlying structure of the picture.

References

[EF01] Alexei A Efros and William T Freeman. "Image quilting for texture synthesis and transfer". In: (2001), pp. 341–346.

[Faz02] Maryam Fazel. "Matrix rank minimization with applications". In: *Elec Eng Dept Stanford University* 54 (2002), pp. 1–130.

[Kwa+03] Vivek Kwatra et al. "Graphcut textures: image and video synthesis using graph cuts". In: 22.3 (2003), pp. 277–286.

[CPT04] Antonio Criminisi, Patrick Pérez, and Kentaro Toyama. "Region filling and object removal by exemplar-based image inpainting". In: *IEEE Transactions on image processing* 13.9 (2004), pp. 1200–1212.

[SRJ04] Nathan Srebro, Jason Rennie, and Tommi S Jaakkola. "Maximum-margin matrix factorization". In: *Advances in neural information processing systems*. 2004, pp. 1329–1336.

[CR09] Emmanuel J Candès and Benjamin Recht. "Exact matrix completion via convex optimization". In: *Foundations of Computational Mathematics* 9.6 (2009), pp. 717–772.

[MHT09] Rahul Mazumder, Trevor Hastie, and Rob Tibshirani. "Regularization methods for learning incomplete matrices". In: *arXiv preprint arXiv:0906.2034* (2009).

[CCS10] Jian-Feng Cai, Emmanuel J Candès, and Zuowei Shen. "A singular value thresholding algorithm for matrix completion". In: *SIAM Journal on Optimization* 20.4 (2010), pp. 1956–1982.

[MHT10] Rahul Mazumder, Trevor Hastie, and Robert Tibshirani. "Spectral regularization algorithms for learning large incomplete matrices". In: *Journal of machine learning research* 11.Aug (2010), pp. 2287–2322.

[Has+15] Trevor Hastie et al. "Matrix completion and low-rank SVD via fast alternating least squares". In: *Journal of Machine Learning Research* 16 (2015), pp. 3367–3402.

Figure 12: Original image, incomplete image and filled image

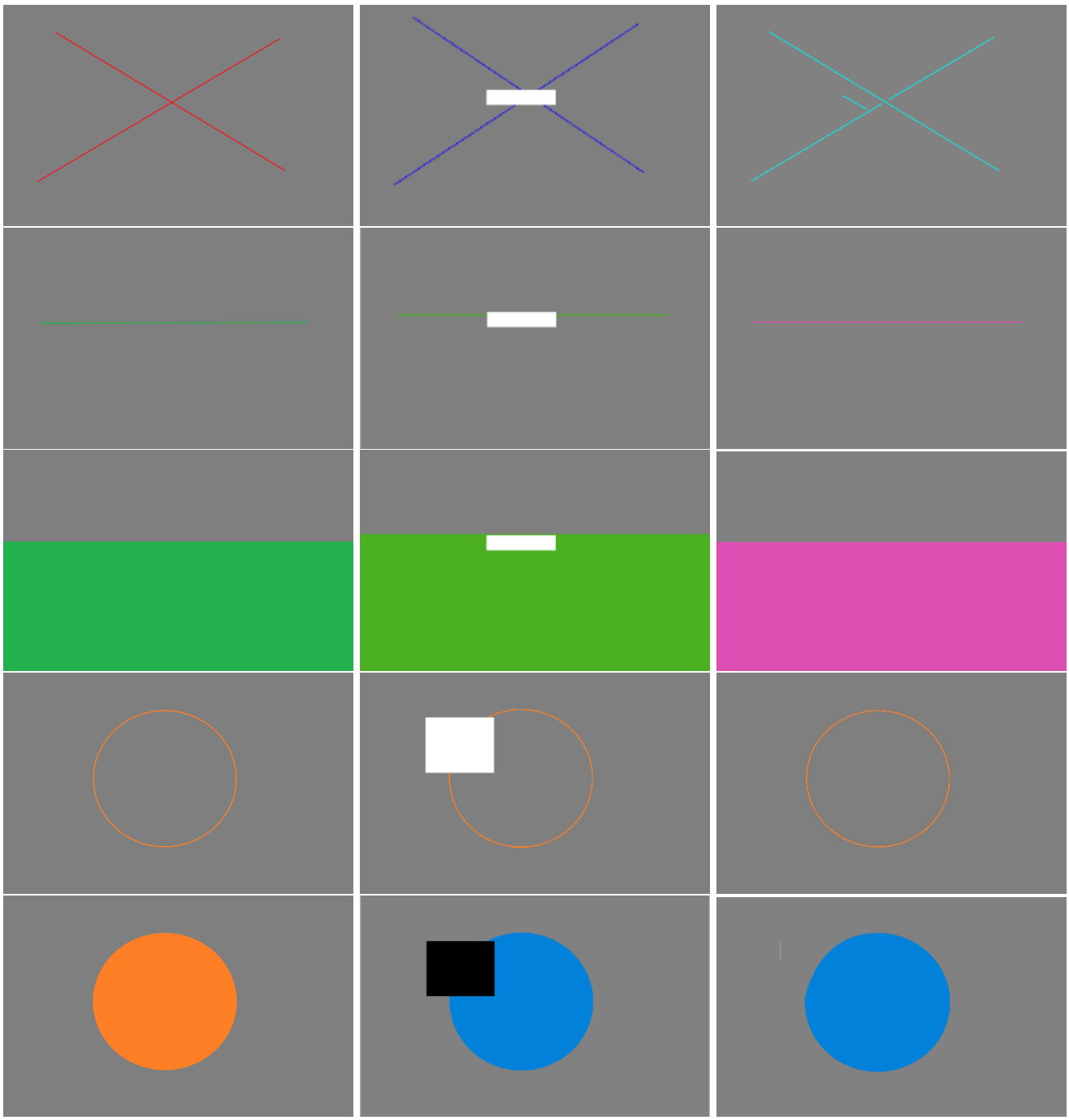


Figure 13: Result of Patch Match with a bungee jumper

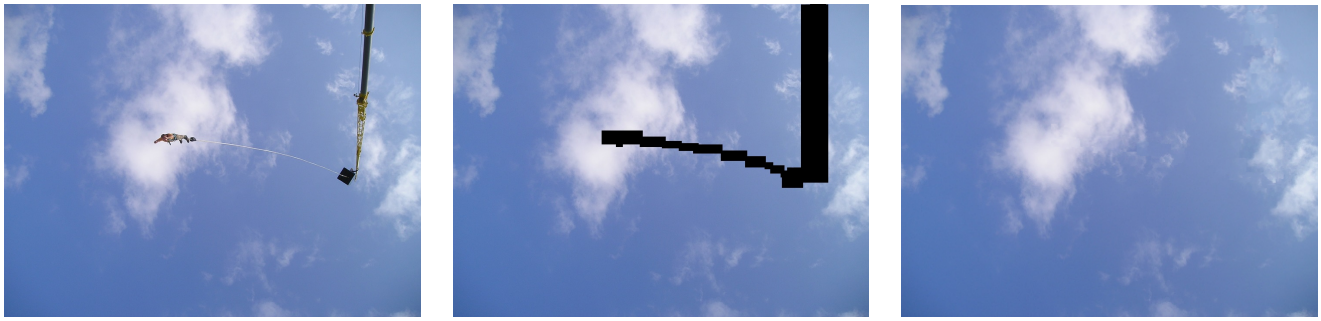


Figure 14: Result of Patch Match with Jerusalem Temple

Figure 15: Result of Patch Match with a skier