Projet ARF: Inpainting

Urien Loïc et Benlarbi Samy

Abstract

Dans le cadre du projet d'ARF, nous avons mis en place l'algorithme du LASSO dans le cadre de l'*inpainting*. Après un préambule autour des différents types de régression, nous présenterons les détails d'implémentation, puis les résultats concernant différentes paramétrisations du problème (bruit,...) et enfin, nous proposerons des heuristiques pour répondre à la dernière partie faisant appel à la lecture de l'article numéro 3 du sujet.

Contents

1 Préambule

1.1 Principe des algorithmes

Dans le cadre de la classification, on rencontre dans de nombreux cas le principe de régression linéaire. Plus précisément , cela revient optimiser une fonction coût par une descente de gradient. Considérons un ensemble d'apprentissage $E = \{(x_i, \hat{y_i}) \in \mathbb{R}^d \times \mathbb{R}\}$. On cherche à minimiser la fonction, avec i $\in \{1, 2\}$:

$$L_i(f_w, E) = MSE(f_w, E) + \alpha \|w\|_i^i$$

où $MSE(f_w, E) = \frac{1}{n} \sum_{i=0}^{n} (f_w(x_i) - \hat{y})^2$, l'erreur aux moindres carrés.

La régression linéaire s'attarde sur la minimisation de cette MSE. Suivant la valeur de i, on aura par contre affaire à deux autres types de régression:

- si i = 1, cela correspond à la régression dite **Lasso**
- si i= 2, cela correspond à la régression dite **Ridge**

Dans le cadre de cette première partie, on utilisera l'implémentation de la librairie *sklearn* pour mesurer plusieurs éléments selon la valeur du α d'une part pour la régression linéaire classique, mais aussi pour les deux autres types rencontrés.

1.2 Résultats

Voici le tableau récapitulatif des différentes valeurs:

		Nombre de composantes nulles	Score	Moyenne des composantes
Linéaire	$\alpha = 1.00$	0	0.589	0.0088
Ridge	$\alpha = 0.25$	0	0.5890	0.0087
	$\alpha = 0.5$	0	0.5891	0.0086
	$\alpha = 0.75$	0	0.5891	0.0085
	$\alpha = 1.00$	0	0.5891	0.0084
Lasso	$\alpha = 0.25$	228	0.491	0.002237
	$\alpha = 0.5$	241	0.394	0.002634
	$\alpha = 0.75$	248	0.291	0.002060
	$\alpha = 1.00$	252	0.190	0.001185

On remarque premièrement, qu'en passant au LASSO, le nombre de composantes nulles du vecteur de poids obtenu augmente drastiquement, alors qu'il était nul avec les autres régressions. Les vecteurs obtenus après les régressions LASSO sont donc très "sparses" (sur 256 composantes, on commence de suite à 228 valeurs nulles). On observe également que plus la valeur de α augmente, plus la moyenne des composantes des vecteurs de poids diminue et semble converger vers 0, surtout pour LASSO, mais avec un score un peu moindre. On peut alors voir que LASSO y trouve son utilité: en amenant des dictionnaires faibles et donc un stockage moins problématique de l'information. La méthode en devient plus intéressante, puisque seules les caractéristiques "utiles" du vecteur de poids sont conservées.

2 LASSO et Inpainting

2.1 Implémentation

Les méthodes proposées dans le sujet ont été implémentés (question 2.1, 2.2) mais certaines choses ont été changé:

- aucune fonction de conversions de patchs en vecteurs n'a été faite puisqu'on l'a fait directement au sein des autres méthodes, certaines opérations de numpy effectuant automatiquement ces opérations.
- nous avons utilisé la régression Lasso de sklearn, et plus précisément le LassoCV pour la cross validation. Nous trouvions qu'il n'était pas très utile de trouver le paramètre λ à la main, alors que la cross validation permettait d'en trouver un optimal assez rapidement.

  En ce qui concerne le paramètre λ, bien que dans la suite du projet LassoCV a été utilisé, nous avons conduit quelques expérimentations afin de s'assurer de son effet sur le résultat. Nous avons ainsi pu nous rendre compte que si le λ est pris trop grand, le résultat final a tendance à être nul ou proche du vecteur nul. S'il est trop petit ceci étant, le résultat a tendance à s'approcher d'une simple régression linéaire. Dans la littérature, il est explicité que ce paramètre de pénalisation est nécessaire pour garantir la "sparsity" du résultat final.

L'algorithme de denoising est relativement simple, nous avons utilisé la fonction LassoCV de sklearn et nous l'avons utilisé afin d'effectuer la régression comme indiqué dans le sujet. En ce qui concerne le dictionnaire des atomes, nous avons effectué une petite heuristique qui a permis d'améliorer le temps de calcul et la précision de la reconstruction dans certains cas :

- Nous construisons le dictionnaire ψ entier des patchs complets et nous le gardons en mémoire
- A chaque patch incomplet, nous utilisons un paramètre "rayon" qui permet d'indiquer un rayon sur lequel nous allons rechercher les patchs environnants du patch que nous sommes actuellement en train d'étudier et nous récupérons les patchs environnants dans le dictionnaire partiel ψ'

- S'il existe des patchs environnants et qu'il sont assez nombreux, alors nous utilisons ψ', sinon nous utilisons le dictionnaire complet ψ

  Nous calculons ensuite le patch complété et nous remplaçons le patch original incomplet par le nouveau patch calculé. Nous suivons un ordre bien particulier afin de compléter l'image, nous effectuons en effet un parcours circulaire des pixels ce qui permet d'implémenter non seulement le denoising mais aussi l'inpainting.

2.2 Résultats des tests

Dans cette section nous montrons les résultats obtenus par notre méthode en utilisant plusieurs images d'exemples et paramètres. Comme indiqué dans le sujet, nous nous sommes efforcé de prendre des images de type texture très cohérentes, et bien que nous ayons effectué des tests sur des images plus complexes les résultats s'en sont trouvés moins convaincants. Nous commencerons donc par montrer les capacités de denoising de notre algorithme avant de passer aux capacités d'inpainting.

2.2.1 Résultats de denoising

Nous présentons ici les résultats de denoising de notre algorithme. Pour tous les tests, nous avons utilisé un "rayon d'action" (voir l'explication de l'implémentation) de 3 pour les petites images et de 5 pour des images plus grandes, avec un stepping de 3. Voici quelques exemples de résultats sur texture cohérentes :

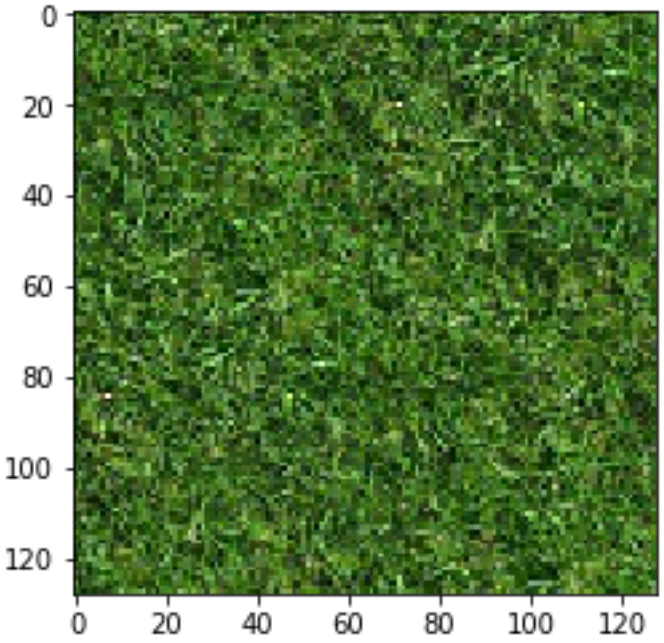

Figure 1: Image "grass" originale

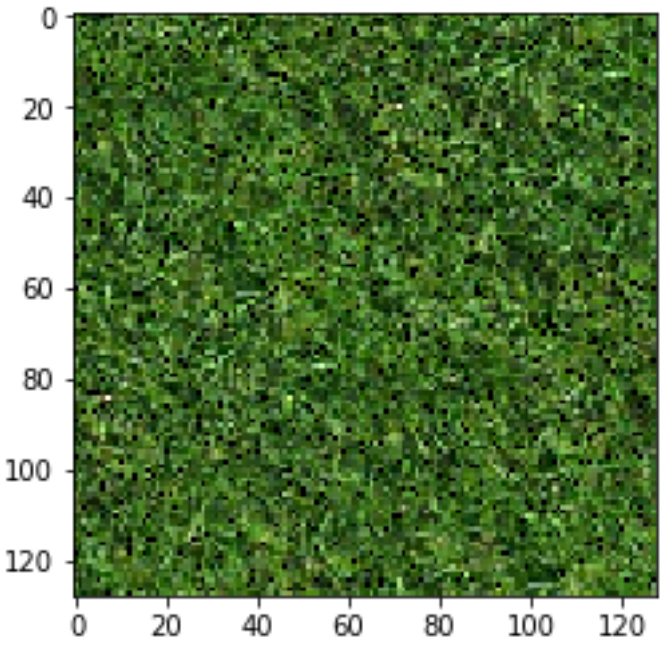

Figure 2: Image "grass" avec 10% de bruit

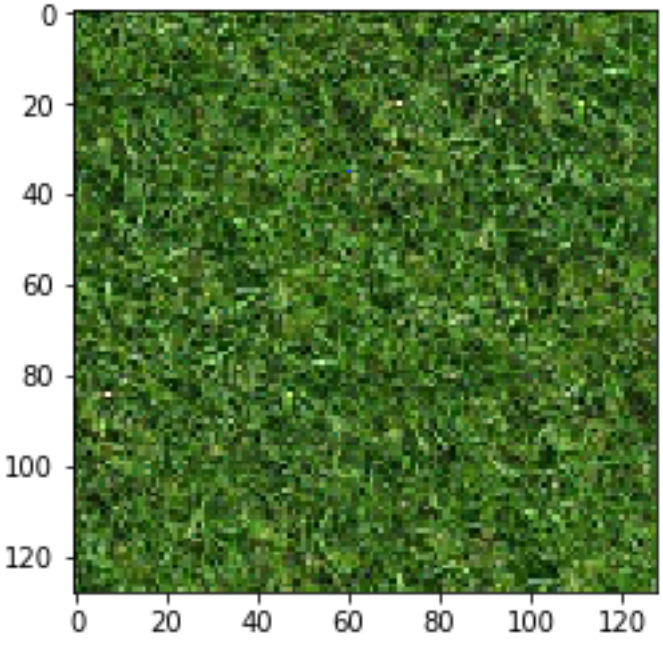

Figure 3: Image "grass" reconstruite à partir de l'image bruitée à 10%

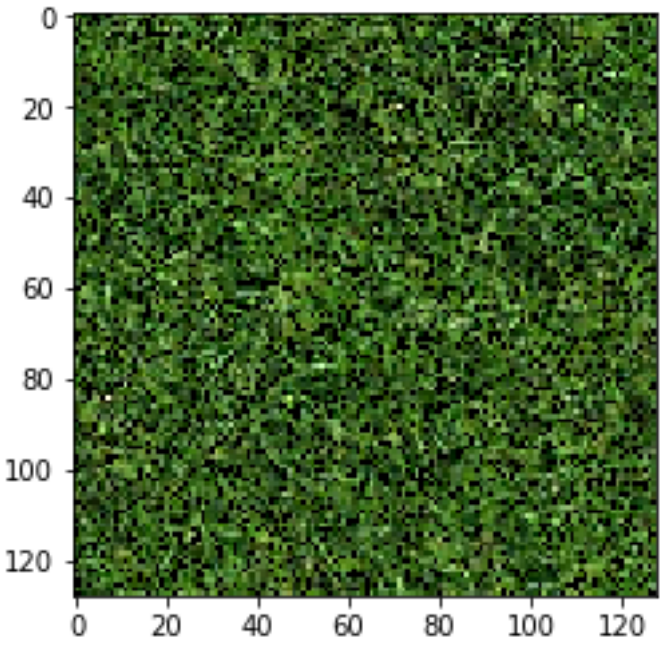

Figure 4: Image "grass" avec 50% de bruit

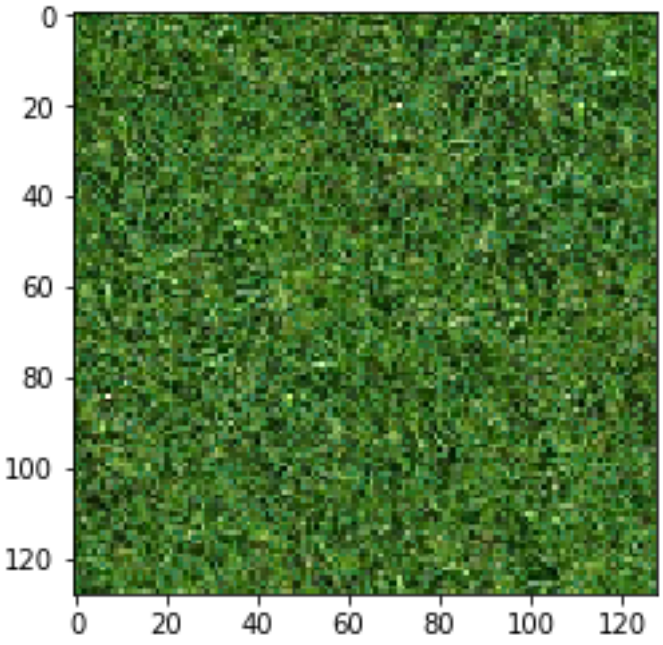

Figure 5: Image "grass" reconstruite à partir de l'image bruitée à 50%

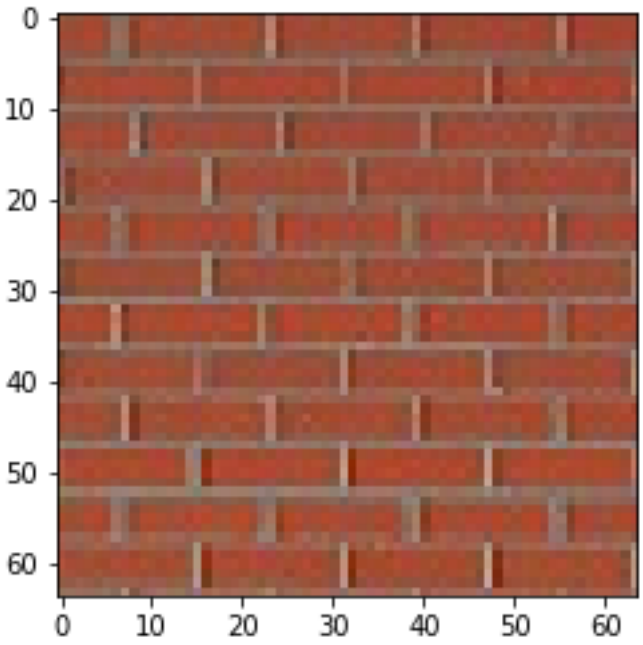

Figure 6: Image "brick" originale

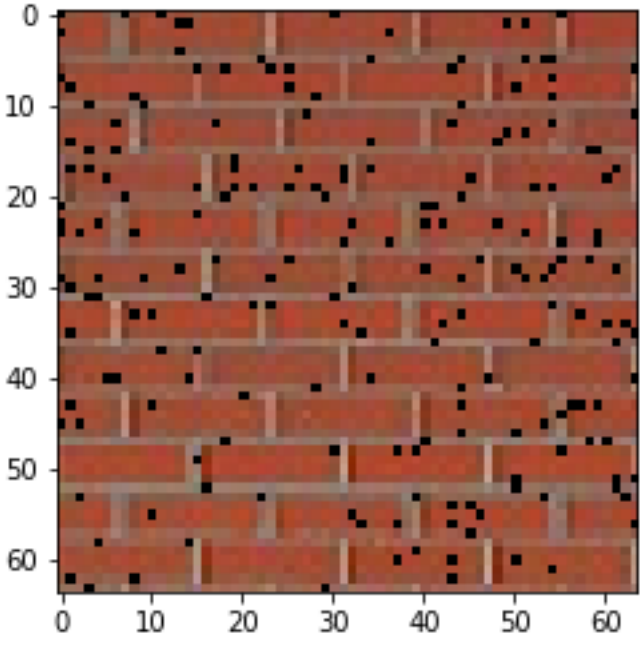

Figure 7: Image "brick" avec 10% de bruit

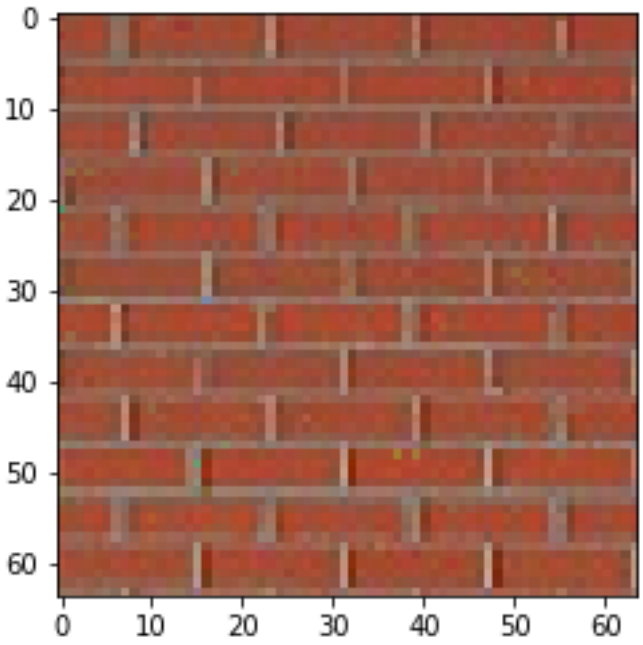

Figure 8: Image "brick" reconstruite à partir de l'image bruitée à 10%

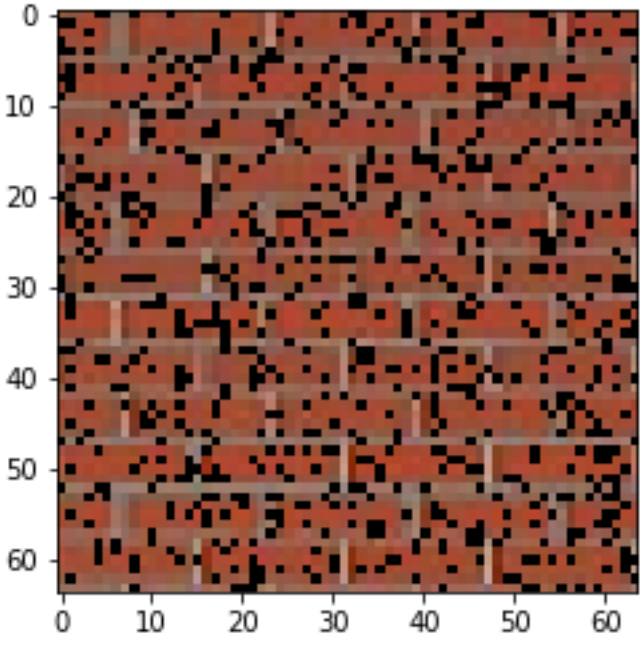

Figure 9: Image "brick" avec 20% de bruit

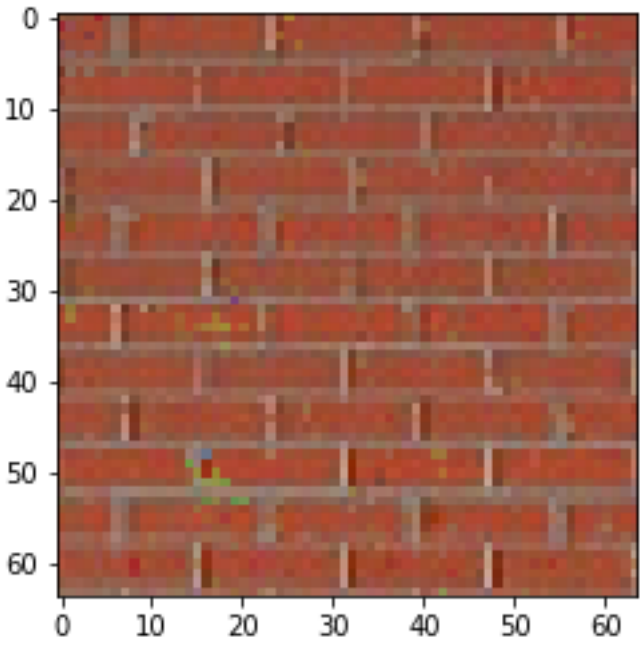

Figure 10: Image "brick" reconstruite à partir de l'image bruitée à 20%

L'algorithme donne de bons résultats sur des images de type textures, la reconstruction est globalement convaincante même avec un haut taux de bruit. Nous avons également essayé d'utiliser l'algorithme sur des images plus complexes afin d'en évaluer les performances. Voici un exemple d'image, le bien connu fond d'écran de Windows XP :

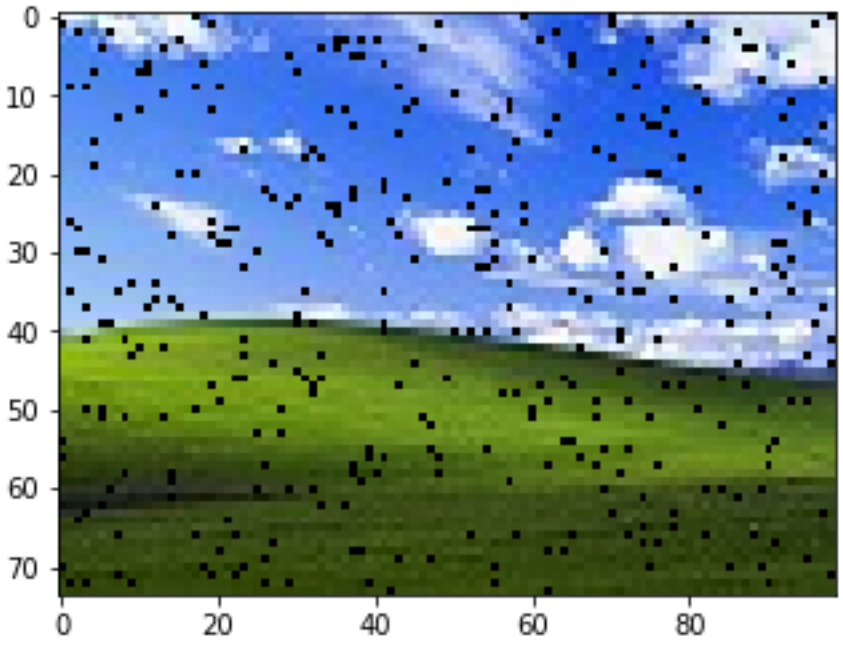

Figure 11: Image "windows" avec 5% de bruit

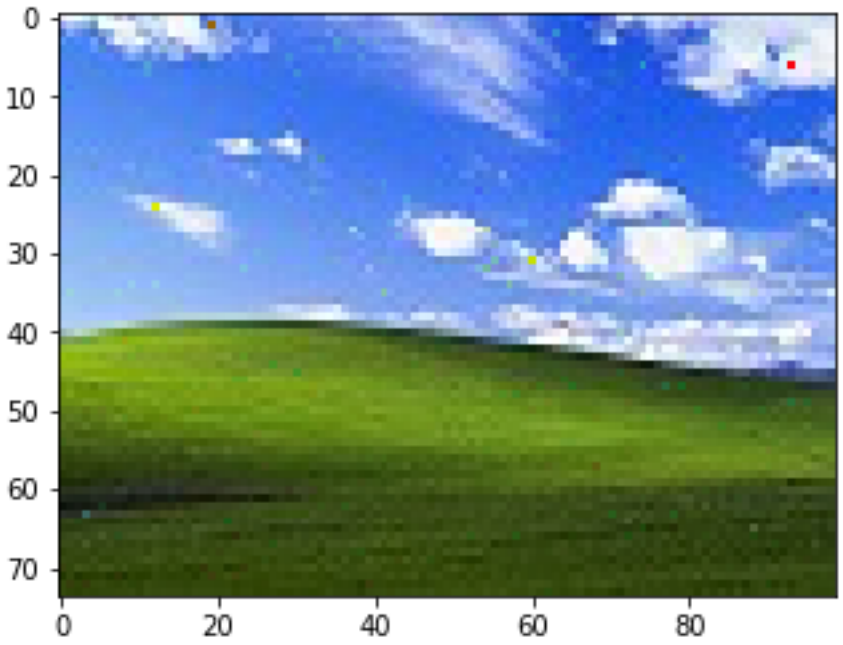

Figure 12: Image "windows" reconstruite à partir de l'image bruitée à 5%

La reconstruction s'avère cette fois bien moins convaincante, bien que l'image ne contenait que 5% de bruit, on peut noter quelques artefacts comme le pixel rouge en haut à droite de l'image.

2.2.2 Inpainting

Dans cette section, nous présentons les performances de l'algorithme sur le problème de l'inpainting. Nous remarquerons que plus la taille de la zone manquante augmente, plus les problèmes de "smearing" et d'imprécision augmentent. Voici quelques exemples de résultats :

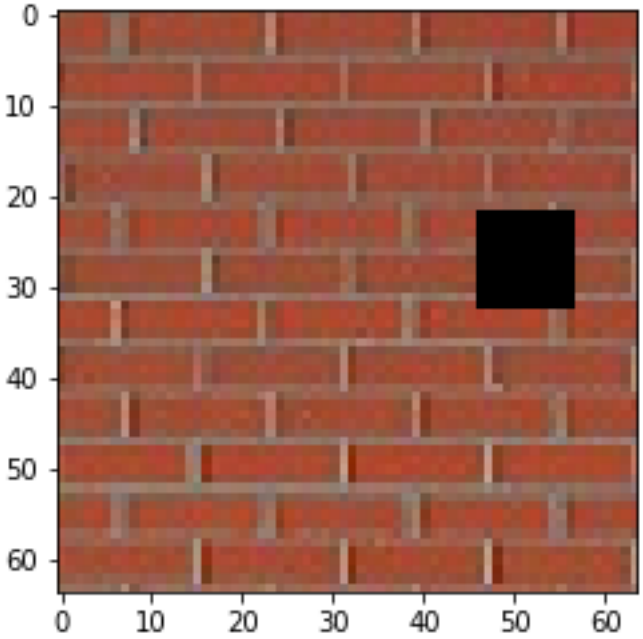

Figure 13: Image "brick" avec un carré de côté 10 manquant

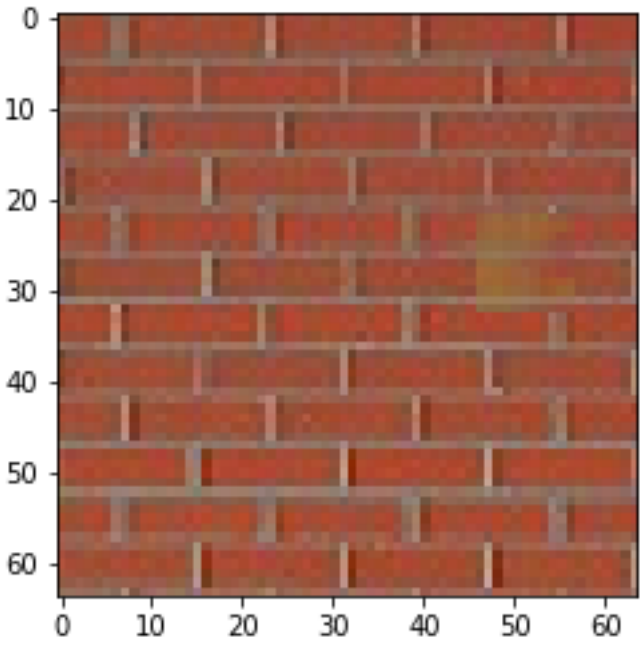

Figure 14: Image "brick" reconstruite ç partir de l'image avec un carré de côté 10 manquant

On peut observer que la structure des briques est bien conservée bien qu'elle soit très légèrement floutée et décolorée par l'algorithme d'inpainting. En essayant une zone plus grande ceci étant, les résultats sont moins bons

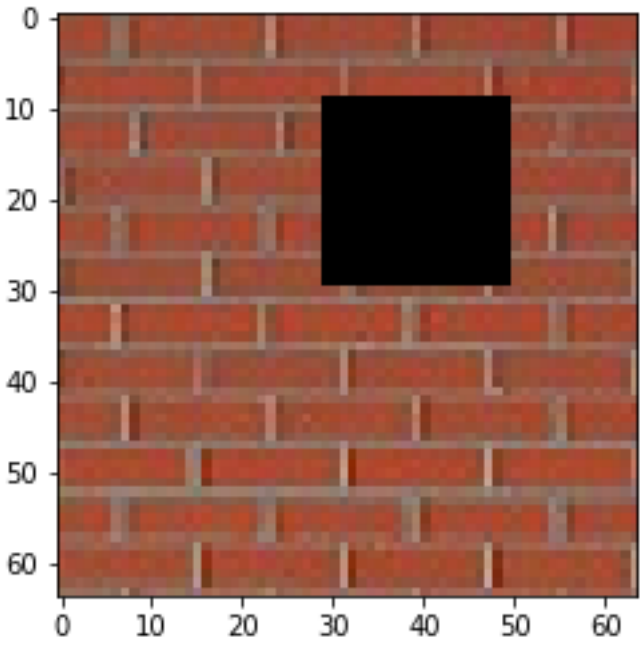

Figure 15: Image "brick" avec un carré de côté 20 manquant

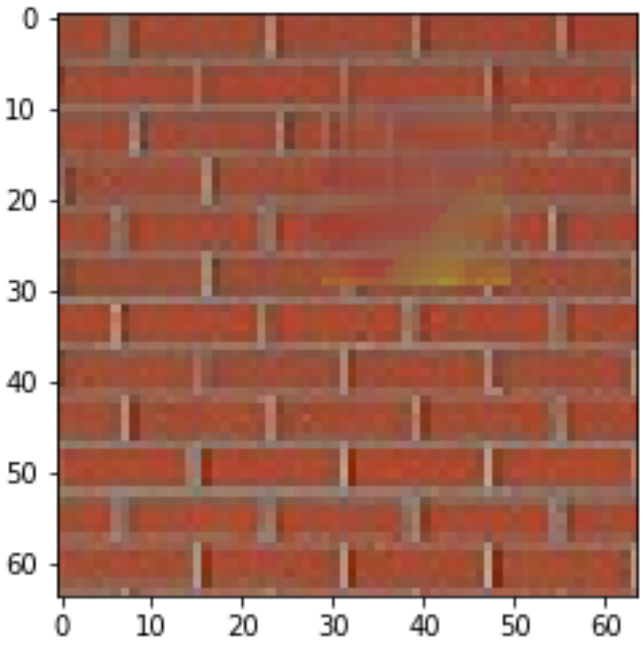

Figure 16: Image "brick" reconstruite à partir de l'image avec un carré de côté 20 manquant

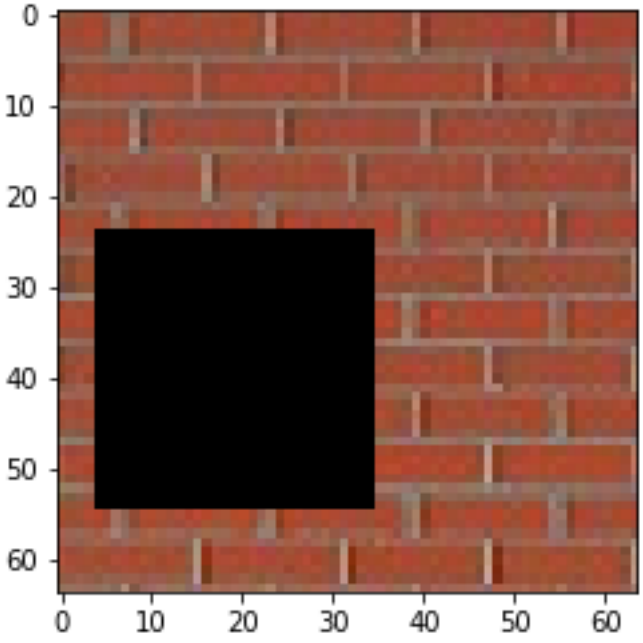

Figure 17: Image "brick" avec un carré de côté 30 manquant

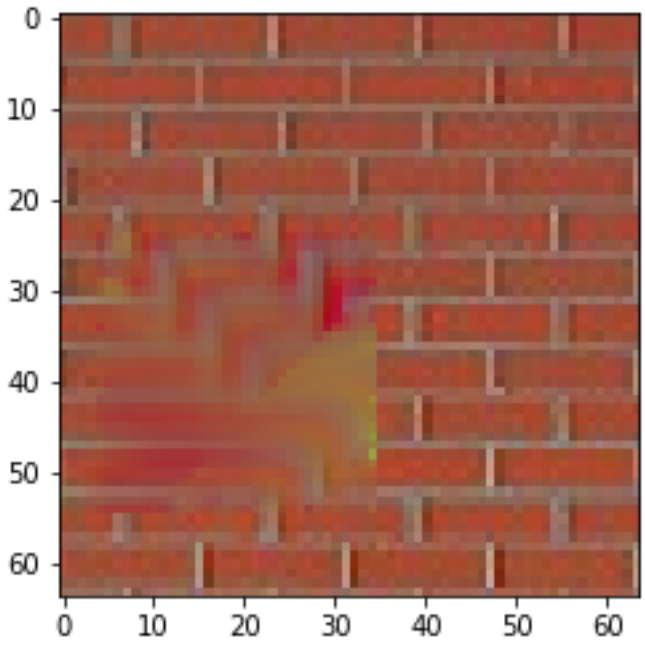

Figure 18: Image "brick" reconstruite à partir de l'image avec un carré de côté 30 manquant

Quand les structures de la texture sont moins fortes ceci étant, les résultats sont bien meilleurs :

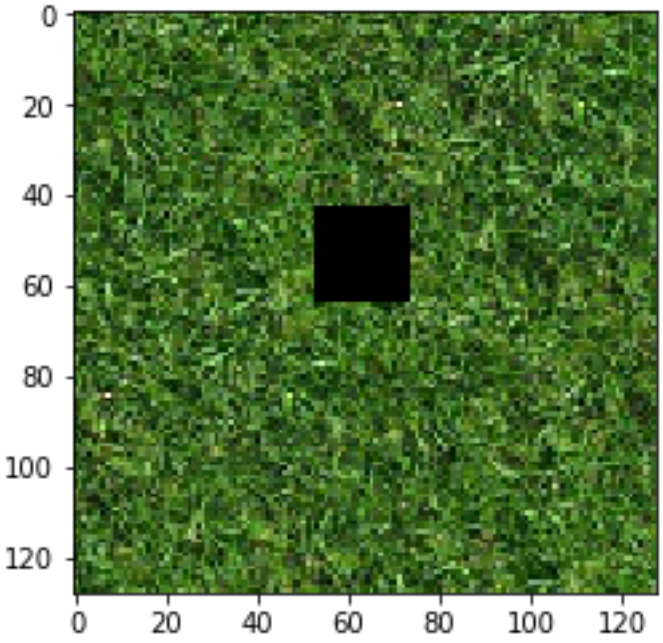

Figure 19: Image "grass" avec un carré de côté 20 manquant

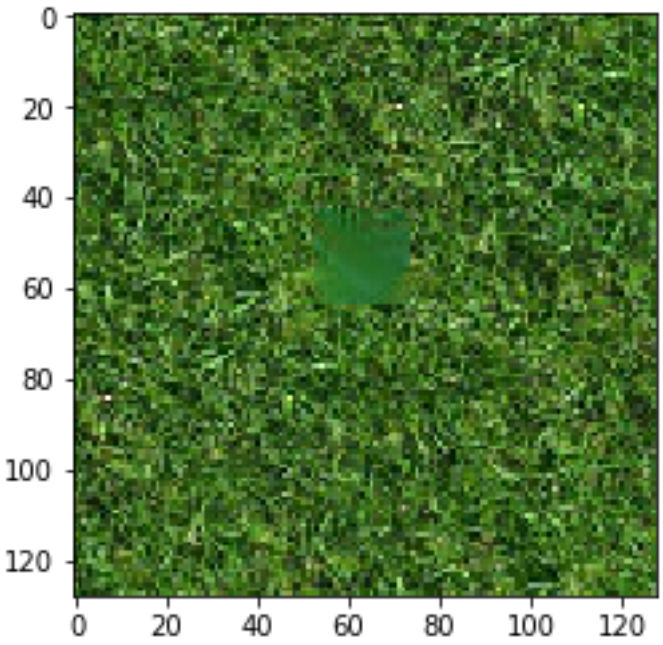

Figure 20: Image "grass" reconstruite à partir de l'image avec un carré de côté 20 manquant

Nous avons également testé notre algorithme sur une image légèrement plus complexe. Nous avons donc réessayé sur le fond d'écran par défaut de Windows XP :

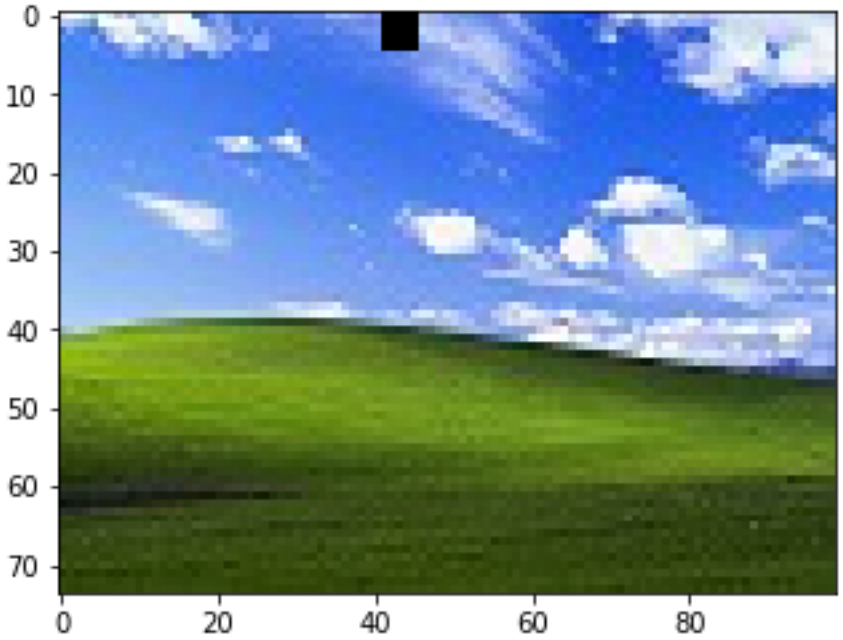

Figure 21: Image "windows" avec un carré de côté 5 manquant

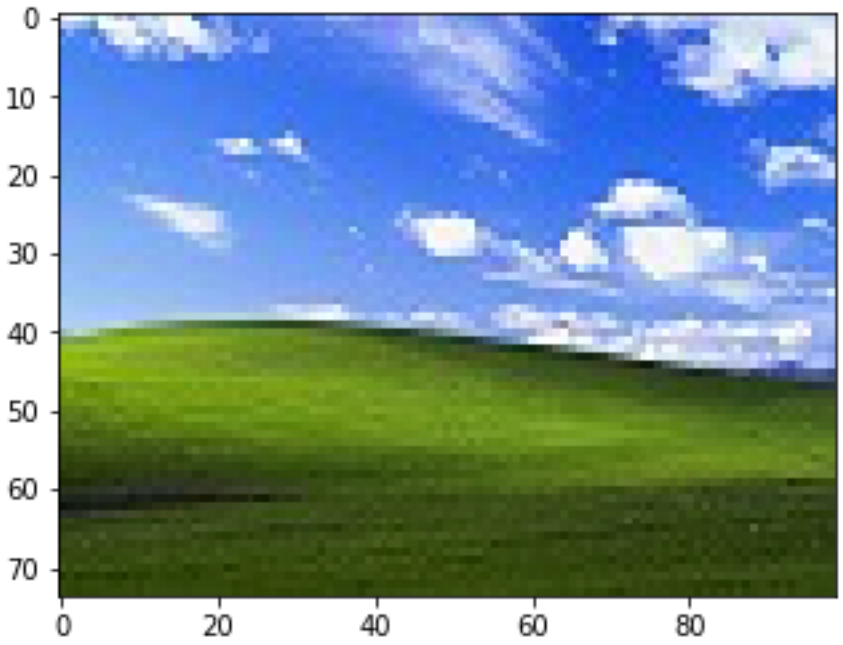

Figure 22: Image "brick" reconstruite à partir de l'image avec un carré de côté 5 manquant

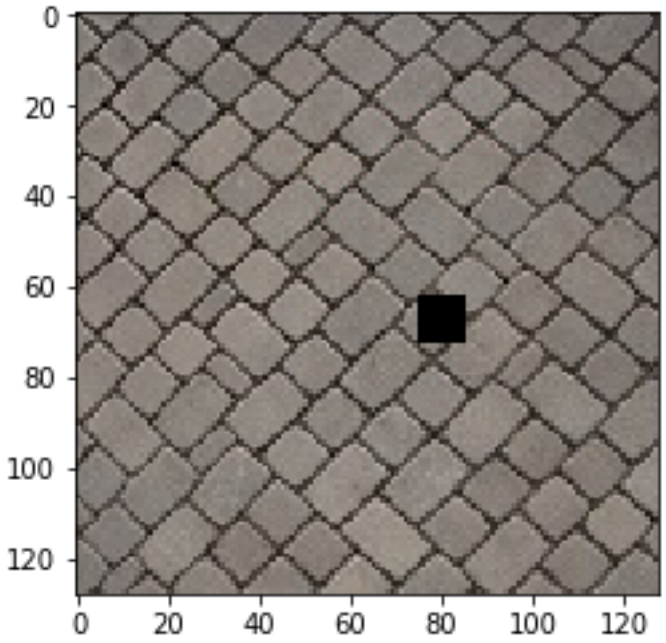

Figure 23: Image "tiles" avec un carré de côté 10 manquant

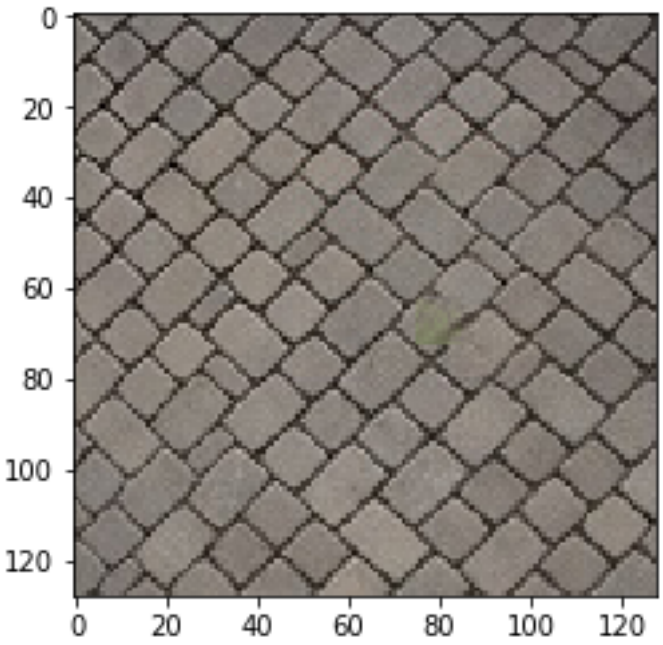

Figure 24: Image "tiles" reconstruite à partir de l'image avec un carré de côté 10 manquant

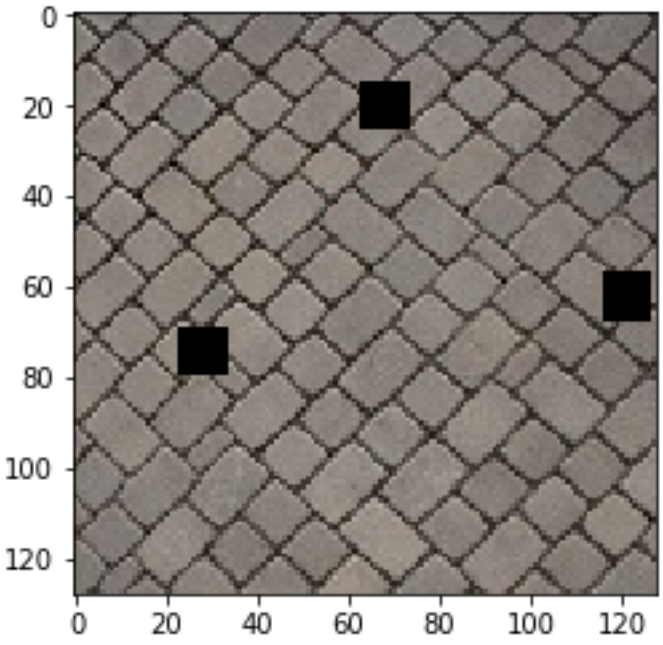

Figure 25: Image "tiles" avec trois carrés de côté 10 manquant

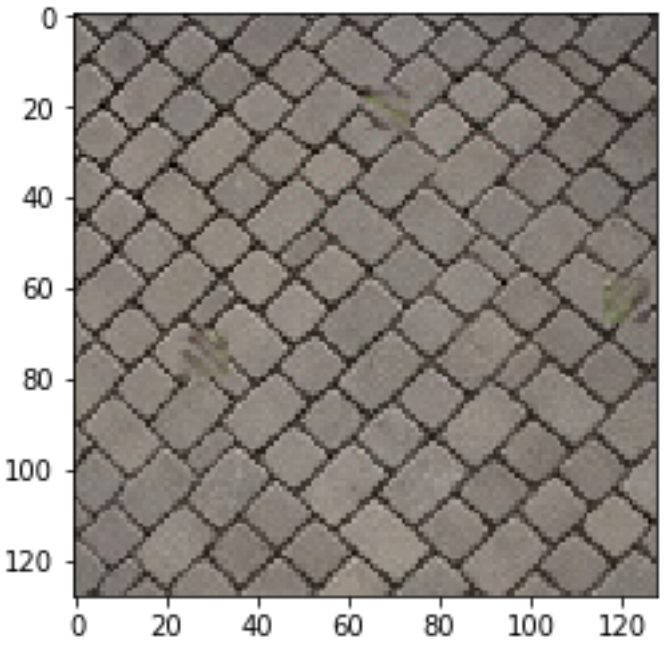

Figure 26: Image "tiles" reconstruite à partir de l'image avec trois carrés de côté 10 manquant

2.3 Pour aller plus loin

Dans le papier 3, il est bien évident que l'ordre de remplissage est extrêmement important dans l'algorithme d'inpainting pour une partie de l'image manquante. La technique la plus simple et appelée "onion peel", ou "épluchage d'oignon" en français. Cette technique consiste à partir des pixels hors de la zone manquante et de petit à petit remplacer les pixels de ce contour de manière circulaire. Dans le papier 3, *Region Filling and Object Removal by Exemplar-Based Image Inpainting*, la section **II.B** intitulée "Filling order is critical" donne un bref aperçu des problèmes liés à ce remplissage sommaire et sans heuristiques. Le cœur de ce papier est de donner un meilleur algorithme de remplissage en utilisant la même méthode d'inpainting que celle employée dans le projet en donnant cette fois des valeurs de priorités aux pixels manquants à remplir. L'avantage de la technique "onion peel" est qu'elle est beaucoup plus simple à implémenter, la preuve étant qu'il a été possible de l'implémenter dans le projet sans grande difficulté, la technique présentée ici étant en effet bien plus compliquée. Nous pouvons résumer simplement la technique comme suit :

- A chaque étape de la reconstruction, on calcule pour chaque patch centré en $\mathbf{p}$ une priorité de la forme $P(\mathbf{p}) = C(\mathbf{p})D(\mathbf{p})$
- On a $C(\mathbf{p}) = \frac{\sum_{\mathbf{q}\in\psi_p\cap\phi} C(\mathbf{q})}{|\psi_p|}$ où ϕ est l'image dont on a soustrait la zone à supprimer (dans notre cas, comme la zone est manquante nous avons déjà ϕ), en d'autres terme c'est la moyenne des $C(\mathbf{q})$ dans notre patch centré en $\mathbf{q}$
- On a $D(\mathbf{p}) = \frac{\delta I_p . n_p}{\alpha}$ où α est un terme régulateur, δI_p est le gradient sur le patch et n_p le vecteur normal à la zone à enlever. Le gradient est calculé en utilisant un simple filtre de Sobel par exemple, puis en prenant le max dans le matrice résultante, et le vecteur normal est calculé comme le vecteur normal au vecteur entre deux points de contrôle de la zone (les points de contrôles sont calculés en utilisant une filtre gaussien bidimensionnel).
- On choisit ensuite le pixel manquant ayant la plus grande valeur de priorité et on le remplit.

Selon les auteurs, cette technique permettrait de conserver les structures et d'éviter d'avoir trop de "smearing" ou création de flou lors de la propagation de la correction. Cependant, après avoir effectué quelques recherches, il semblerait que cette technique soit relativement peu utilisée aujourd'hui, en partie dû au fait que les réseaux de neurones à convolution semblent globalement donner des meilleurs résultats pour ce genre de problèmes.