Projet Machine Learning : Image Inpainting

Jean Soler, Nicolas Castanet
Sorbonne Université
Enseignant : Nicolas Baskiotis

Juin 2020

I. Introduction

Pour réaliser ce projet d'inpainting d'images, nous nous sommes inspiré des articles de A. Criminisi et de Bin Shen. En effet, d'une part, nous utilisons l'algorithme du LASSO afin d'approximer les patch à reconstituer, d'autre part, nous calculons à chaque itérations un ordre de priorité des pixels à remplir jusqu'à ce que toute l'image soit reconstituée. Nous avons donc implémenté un algorithme similaire à celui présenté dans le papier de Bin Shen, avec un calcul des priorités des pixels emprunté à l'article de A. Criminisi.

II. Implémentations et test de base

i. Illustration d'une itération

Nous avons commencé par implémenter un certins nombre de fonction utile à l'algorithme d'inpainting :

- Lecture et normalisation de l'image
- Obtenir un patch, ainsi que les indices des pixels le constituant et constituant sa bordure
- Obtenir tous les patch de taille h d'une image
- Construire le dictionnaire de patch intacts
- Approximer un patch avec l'algorithme du LASSO
- Reconstituer un patch bruité avec les coefficients du LASSO
- Bruiter aléatoirement une image
- Supprimer un rectangle d'une image
- Mettre à jour la bordure de la zone à restaurer
- Calculer la priorité des patch à remplir

Nous avons fait des tests de reconstitution de patch bruités après avoir calculé le dictionnaire de patch sur une image Pour une pe-

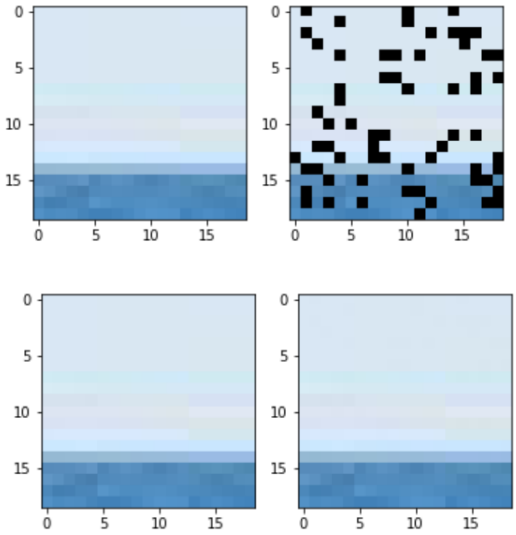

Figure 1: *patch d'origine/patch bruité et patch d'origine/patch débruité*

nalisation du LASSO permettant d'avoir des coefficients non nuls, on est capable de reconstituer presque à l'identique le patch d'origine.

III. Algorithme d'inpainting

Nous illustrons les étape par étape notre algorithme sur une image simple : le drapeau de la suede. La première étape est la sélection d'une zone à restaurer. La bordure de la zone à restaurer est alors calculée. On appel l'image I, la zone à restaurer R et h la taille de coté du patch. Nous prenons ici $h = 20$.

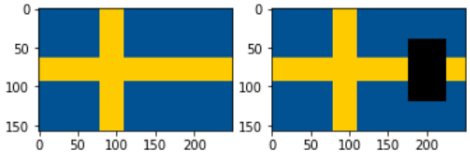

Figure 2: *image d'origine/ image à reconstituer*

L'étape suivante est la création du dictionnaire de patch. Nous prenons pour l'exemple un patch par pixel ce qui donne un total de 31510 patchs dans le dictionnaire.

Une fois le dictionnaire crée, on remplie les patchs des pixels sur la bordure tant qu'il y a des pixels non exprimés. Les pixels à remplir son sélectionné par ordre de priorité. De façon identique au terme de confiance de l'article [2], on calcule le terme $C(p) = \sum_{q \in (I-R)} C(q)/h^2$. On sélectionne donc le pixel qui à l'indice de confiance le plus grand, c'est à dire celui qui possède le plus de pixels de confiance autour de lui. On initialise $C(p) = 1, p \in (I - R)$ et $C(p) = 0, p \in R$.

Nous illustrons une itération de l'algorithme en prenant un pixel au hasard sur la bordure :

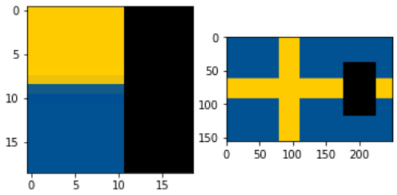

Figure 3: *patch centré en le pixel sélectionné*

L'étape suivante est la reconstitution du patch bruité par l'algorithme du LASSO, ici on prend une pénalisation $\alpha = 0.0005$:

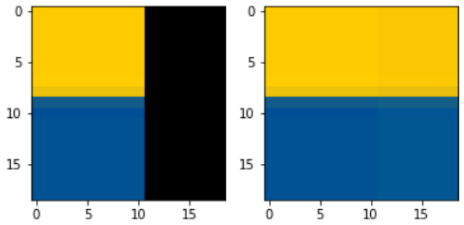

Figure 4: *patch bruité et patch reconstitué*

On remplace ensuite dans I le patch bruité par le patch reconstitué :

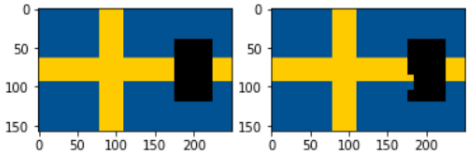

Figure 5: *mise à jour de l'image*

On met ensuite à jour la bordure de la zone à reconstituer et les indices de confiance.

Voici le déroulement de l'algorithme :

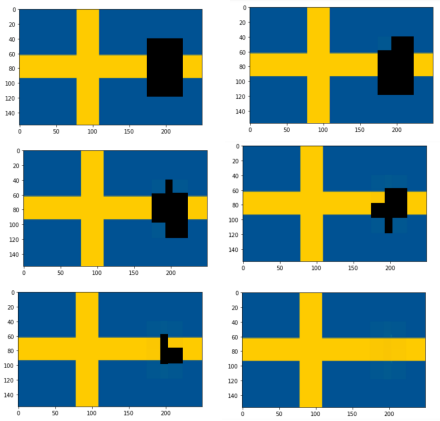

Figure 6: *Reconstitution du drapeau de la Suede*

IV. Impact des hyper parametres

Il y a trois hyper-paramètres dans notre algorithme : le facteur de pénalisation du LASSO α, la taille des patch et le step de sélection des patch. Nous allons étudier leurs impacts dans cette section.

i. Pénalisation du LASSO

La pénalisation du LASSO encourage la diminution du nombre de coefficients non nuls. Ce qui permet de réduire le nombre de patch utilisés pour reconstituer un patch bruité. Nous observons que lorsque $alpha$ est trop élevé, le peu de coefficients non nuls sont presque à zéro ce qui donne une reconstitution médiocre. Plus α est faible, plus il est possible de trouver une reconstitution parcimonieuse. Nous étudions l'impact de α avec $h = 40$ et $step = 1$:

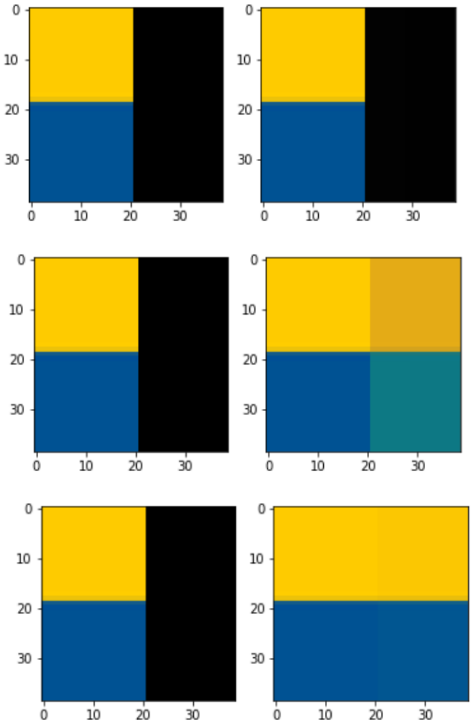

Figure 7: *Reconstitution du patch respectivement pour* $\alpha = 0.1, 0.01, 0.001$

ii. Step de sélection des patchs

Le step impact directement la taille du dictionnaire de patch qui nous servent d'exemples d'apprentissage. Ainsi, plus le step est grand, plus l'exécution est rapide et moins elle est précise. On étudie ici le nombre de patch de taille $h = 40$ dans le dictionnaire pour une image de taille 157x250 :

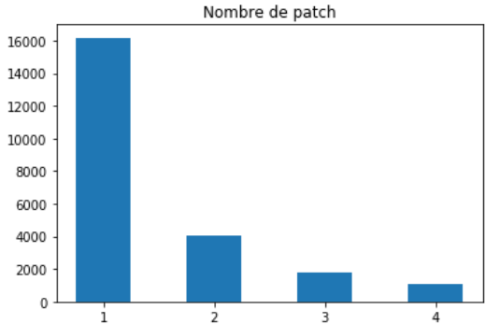

Figure 8: *nombre de patch dans le dictionnaire pour* $step = 1, 2, 3, 4$

On remarque que dans une image nécessitant autant de précision, ce facteur doit être à 1 pour avoir des résultats optimaux :

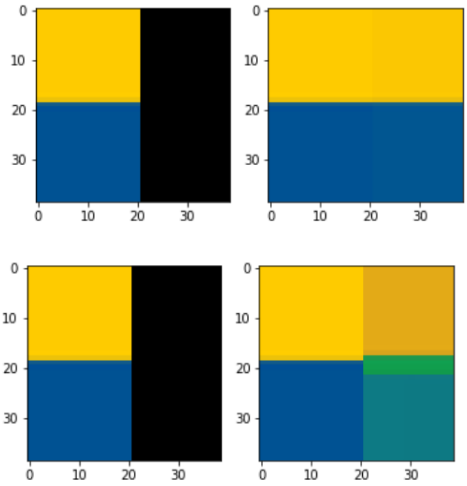

Figure 9: *Reconstitution du patch respectivement pour* $step = 1, 2$ *avec* $\alpha = 0.001$ *et* $h = 40$

iii. Taille du patch

La taille du patch est un hyper-paramètre directement relié à l'image que lon souhaite reconstituer. Il doit être légèrement plus grand que le niveau de texture distinguable dans l'image. Il est à noté que la taille du patch impact directement le temps d'inpainting.

Effectuer une optimisation des hyper-paramètres par GriSearch est ici compliqué car le temps de calcul serait trop long. Il faudrait également optimiser les hyper-paramètres pour chaque image à reconstituer ce qui n'est pas optimal.

V. Résultats avancés et futurs travaux

Nous avons fais plusieurs test de cet algorithme dans la mesure du possible car il est très coûteux en temps (on fait tourner un lasso pour chaque patch) or pour obtenir un résultat correct, il est nécessaire que la taille du patch soit petit, le remplissage des images est donc très long.

i. Résultats

On remarque que les résultats dépendent beaucoup de l'image en elle même et de la taille de la zone à restaurer

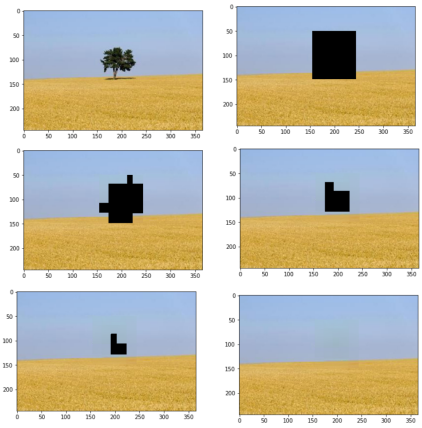

Figure 10: $h = 40$, $step = 2$, $\alpha = 0.001$

On observe ici l'impact de la taille des patchs

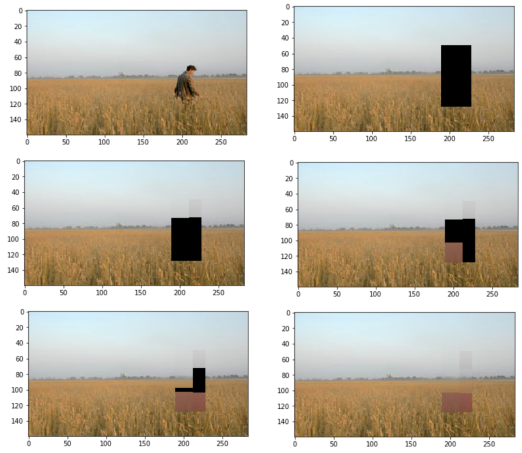

Figure 11: $h = 50$, $step = 2$, $\alpha = 0.0005$

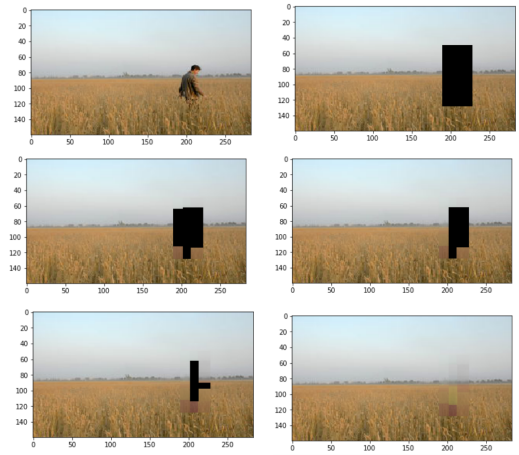

Figure 12: $h = 20$, $step = 2$, $\alpha = 0.0005$

Une campagne d'expérience est difficile à mener du au temps de calcul. On observe que les résultats sont très dépendant de la simplicité de l'image

ii. Futurs travaux

L'optimisation des hyper-paramètres reste encore ici très expérimental, nous gagnerons à la travailler d'avantage, mais le temps de calcul reste un obstacle.

Une composante forte à améliorer serait l'ordre de remplissage. Pour l'instant, nous ne calculons que le niveau de confiance d'un pixel, on pourrait également trouver les points qui sont la continuité d'un contour afin de propager les structures linéaire. De manière générale, il nous reste à optimiser l'implémentation de notre algorithme pour gagner en temps de calcul.