Rapport

Projet: inpainting

(MLL)

Kewei XU

13 Juin, 2020

1 Préambule : régression linéaire, régression ridge et LASSO

1.1 Description

Nous proposons trois modèles: régression linéaire, régression ridge et LASSO adaptées à la classification binaire par la méthode du plug-in. Nous utilisons sklearn comme outil de construction de modèles.

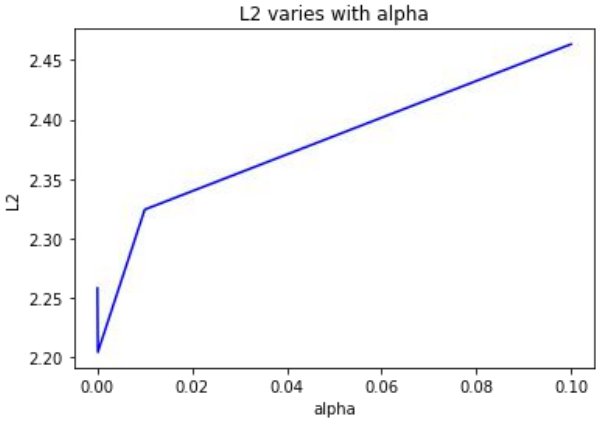
Concernant les indicateurs utilisés pour évaluer les modèles: pour la régression linéaire, nous utilisons MSE (Le carré moyen des erreurs), et pour Ridge et Lasso nous utilisons respectivement L2 et L1. L'ensemble de données utilisé est USPS (l'ensemble de appentissage et l'ensemble de test ont été divisés). Les fonctions de lecture des données du fichier USPS, y compris les fonctions utilisées pour la normalisation, se trouvent dans le fichier ***usps_tools.py***. L'établissement des trois modèles, la prévision des données, l'évaluation des résultats et d'autres fonctions se trouvent dans le fichier ***preambule.py***.

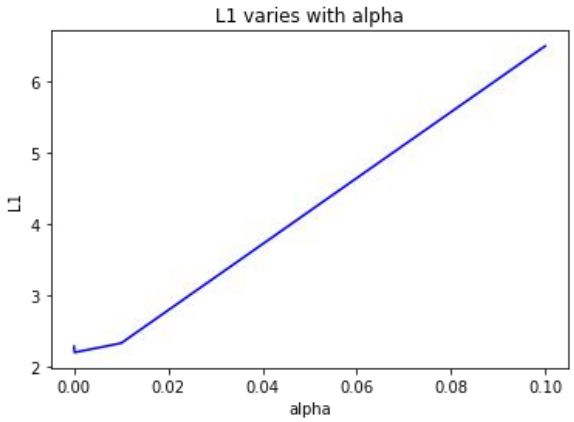
Pour l'évaluation des résultats, nous avons d'abord testé la relation entre alpha et la fonction d'évaluation (qui doit être minimisée). Après cela, un valeur de alpha ont été prises, montrant les poids des coefficients des modèles Régression linéaire, Ridge et Lasso, et le nombre de coefficients null aussi.

1.2 Analyse des résultats

Pénalisation coefficient(alpha) : 0.000001, 0.0001, 0.01, 0.1

Nous avons sélectionné quatre valeurs de alpha avec de grandes portées afin de mieux voir la situation de sous-appentissage et de sur-appentissage. Pour chaque valeur de alpha, nous avons mesuré dix fois et fait la moyenne.

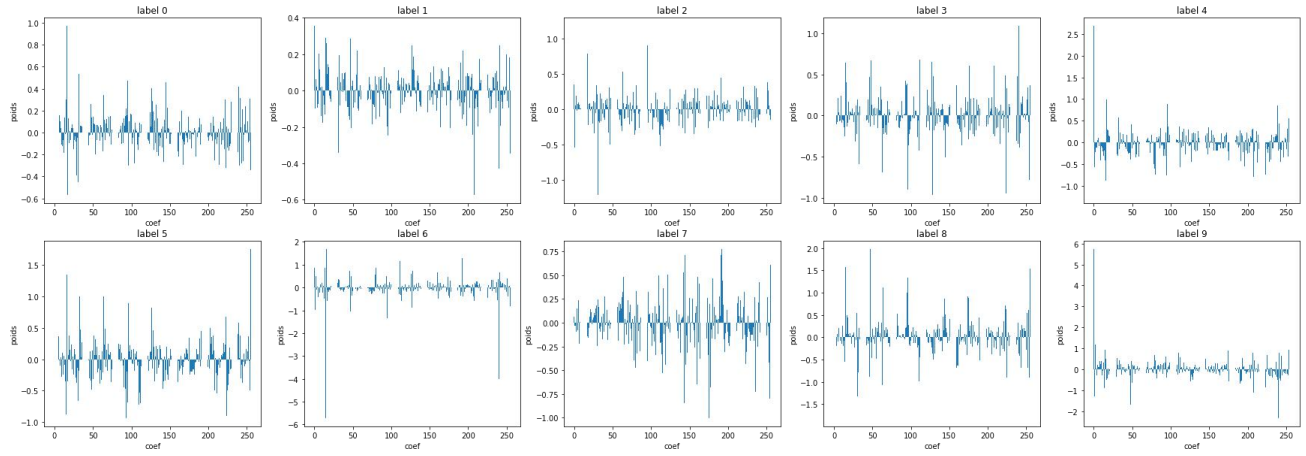




Ridge: L2 score [2.26, 2.20, 2.32, 2.46] **Lasso: L1 score [2.28, 2.20, 2.32, 6.50]**

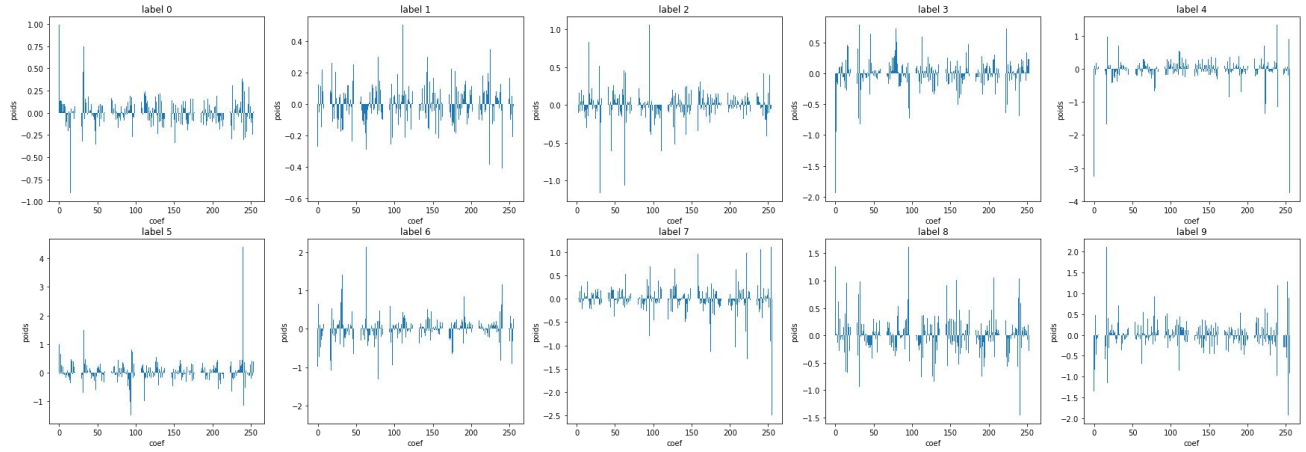
Conclusion: Lorsque alpha est approximativement égal à 0,001, les modèles Ridge et Lasso sont les meilleurs, ni sous-apprentissage ni sur-apprentissage.

Les poids des coefficients de Régression linéaire:



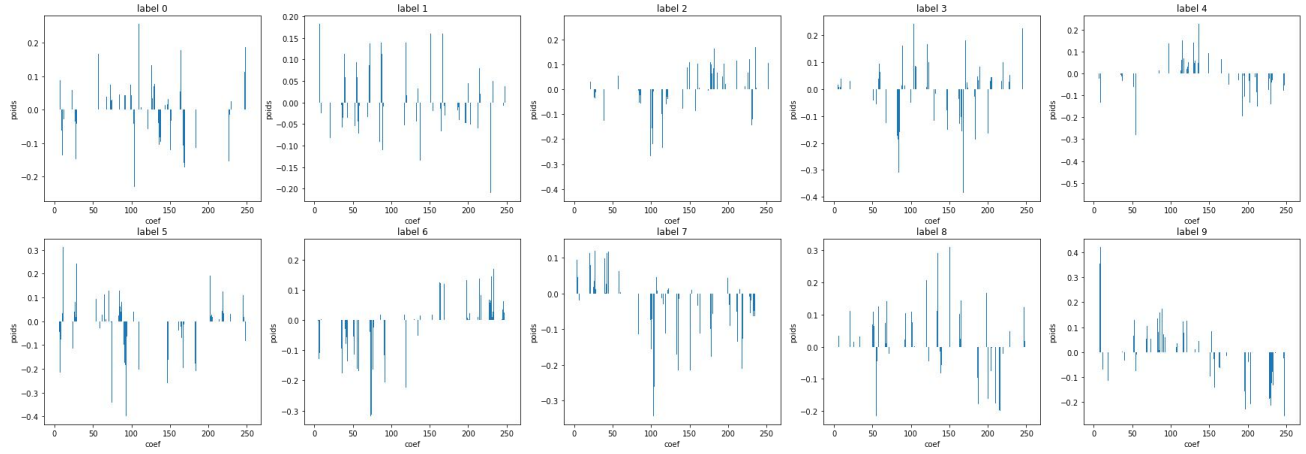
Nombre de coefficients null = 0

Les poids des coefficients de Ridge (alpha = 0.01):



Nombre de coefficients null = 0

Les poids des coefficients de Lasso (alpha = 0.01):



Nombre de coefficients null = 1981

Conclusion:

À partir des trois figures ci-dessus, nous pouvons observer que Ridge et Lasso peuvent réduire les coefficients de dimensions qui ont peu d'effet sur le modèle, mais Ridge ne peut pas éliminer les dimensions et ne peut que continuer à réduire ses coefficients, mais Lasso peut modifier les coefficients à null.

Le lasso peut annuler les dimensions qui ont peu d'effet sur le modèle, de sorte que le modèle n'a besoin de prêter attention qu'à quelques dimensions importantes, mais en même temps, trop de punition pour les dimensions non importantes fera également que certaines dimensions importantes changés a des dimensions non importantes, ce qui cause de la sous-apprentissage.

2 LASSO et Inpainting

2.1 Réparer du bruit d'image

2.1.1 Description

Dans cette partie, nous utiliserons le modèle LASSO pour réparer le bruit sur l'image, lire l'image, afficher l'image, ajouter du bruit et d'autres fonctions sur le fonctionnement de base de l'image peuvent être trouvées dans le fichier ***img_tools.py***, construire le modèle, le patch et d'autres fonctions peuvent être trouvé dans le fichier ***patch_predict.py***. La réparation de l'image complète est implémentée dans le fichier ***noise_repair.py***.

Nous définissons quelques variables dans la fonction:

h: représente la taille du patch

d_h: représente le distance vertical entre les différents patchs

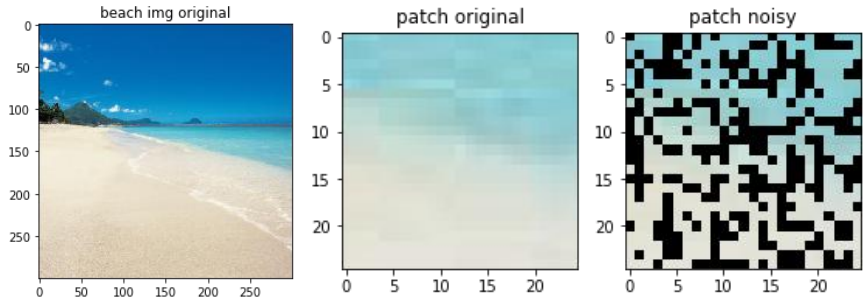
d_w: représente le distance horizontal entre les différents patchs

i, j: représente les coordonnées horizontales et verticales du point central du patch

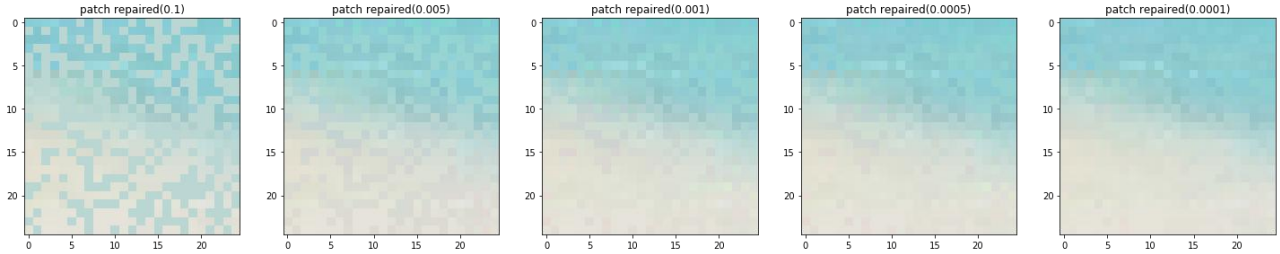
Remarque: La taille du patch doit être un nombre impair, la taille du patch et l'intervalle entre les patchs doivent correspondre à la taille de l'image

2.1.2 Évaluation des résultats

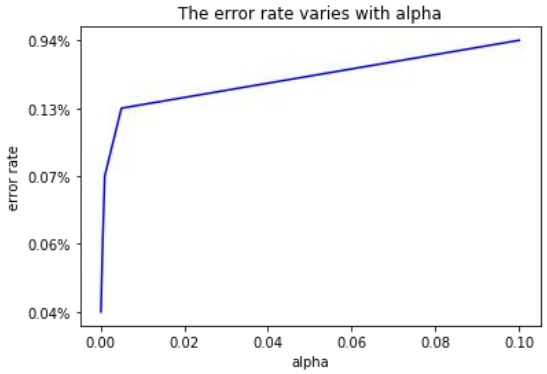
Reparer d'un patch à évaluer en fonction de différentes valeurs alpha(Utiliser l'image complète comme dictionnaire)



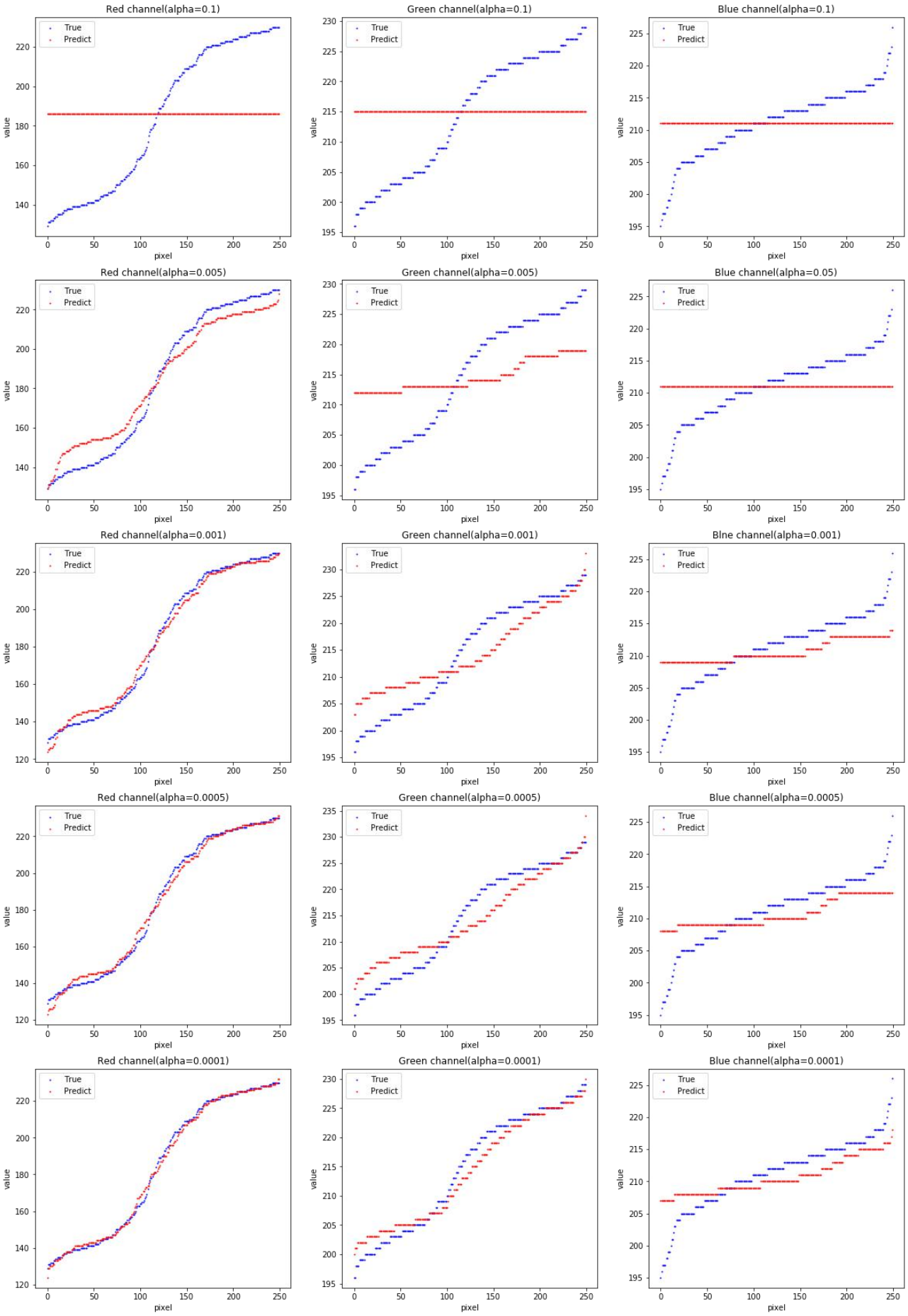
Réparation terminée (entre des parenthèses sont des valeurs de alpha utilisées)



Nous avons calculé la relation entre le taux d'erreur et l'alpha des résultats de la prévision. Comme prévu, une réduction appropriée de la valeur de l'alpha peut réduire le taux d'erreur et éviter un sous-appentissage.



Afin d'observer les résultats de prédiction plus clairement, nous affichons les résultats de prédiction des trois canaux différents sur le graphique, le bleu est la valeur réelle des pixels au bruit et le rouge est le résultat de notre prédiction.

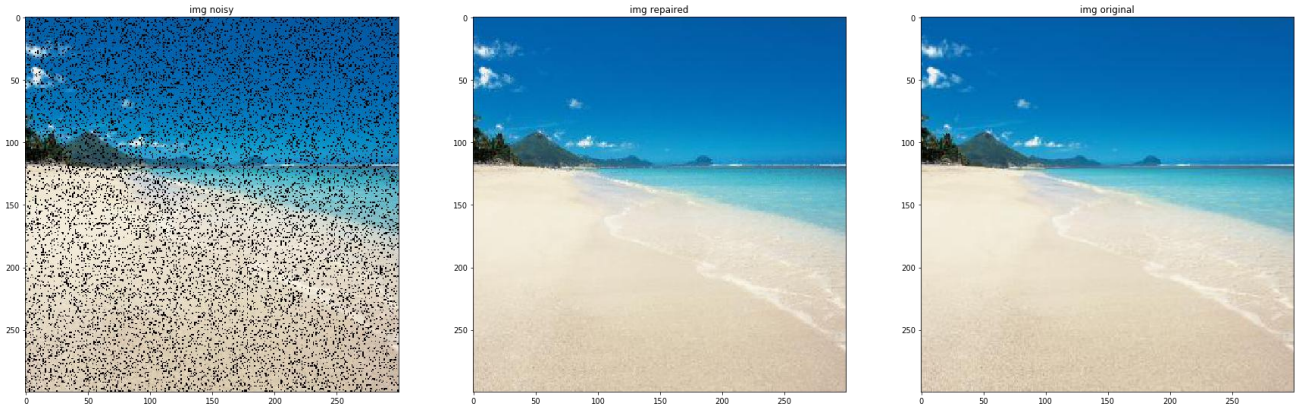

Red channel(alpha=0.1)
Green channel(alpha=0.1)
Blue channel(alpha=0.1)
Red channel(alpha=0.005)
Green channel(alpha=0.005)
Blue channel(alpha=0.05)
Red channel(alpha=0.001)
Green channel(alpha=0.001)
Blne channel(alpha=0.001)
Red channel(alpha=0.0005)
Green channel(alpha=0.0005)
Blue channel(alpha=0.0005)
Red channel(alpha=0.0001)
Green channel(alpha=0.0001)
Blue channel(alpha=0.0001)
True
Predict
value
pixel

Conclusion: Les résultats affichés sont cohérents avec les résultats du taux d'erreur, ce qui signifie que nos prédictions pour alpha sont vraies.

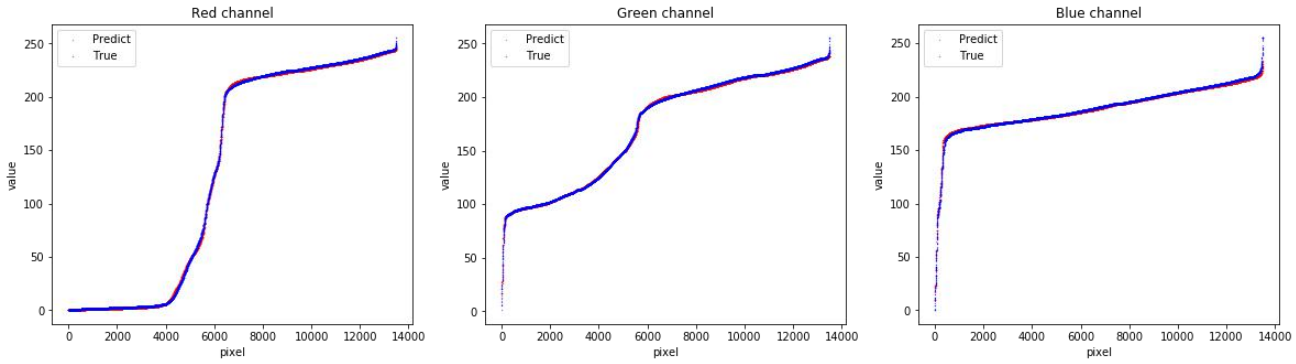
D'après les résultats ci-dessus, nous savons que le modèle que nous avons utilisé pour réparer le bruit est correct, et nous pouvons commencer à réparer l'image entière.

Pour obtenir la dictionnaire qu'on a utilisé est l'image noisy
Les paramètre qu'on a choisi: (alpha = 0.01, la taille du patch = 5, les distances entre patchs = 5)
Temps d'exécution: 7.03s

Réparation terminée (De gauche à droite: image noisy, image repaired, image original)



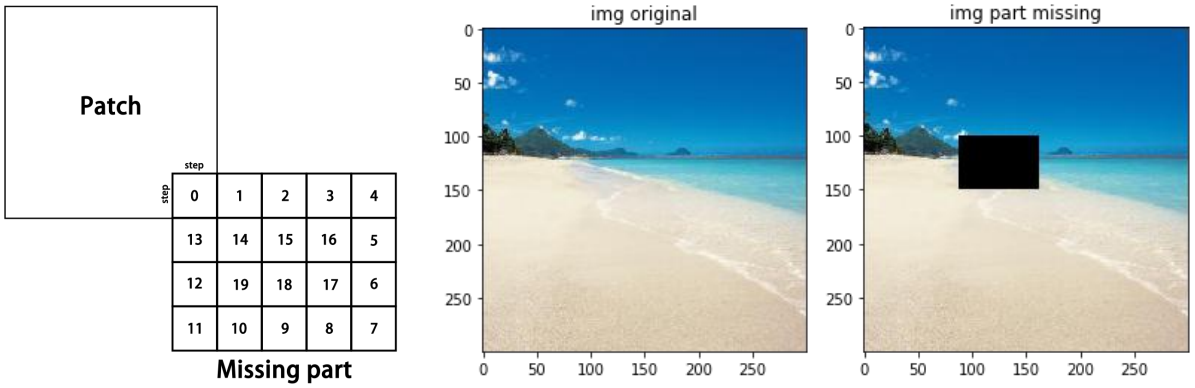
Nous affichons les résultats de prédiction des trois canaux différents sur le graphique, le bleu est la valeur réelle des pixels au bruit et le rouge est le résultat de notre prédiction.



Le résultat est comme prévu, la réparation est réussie.

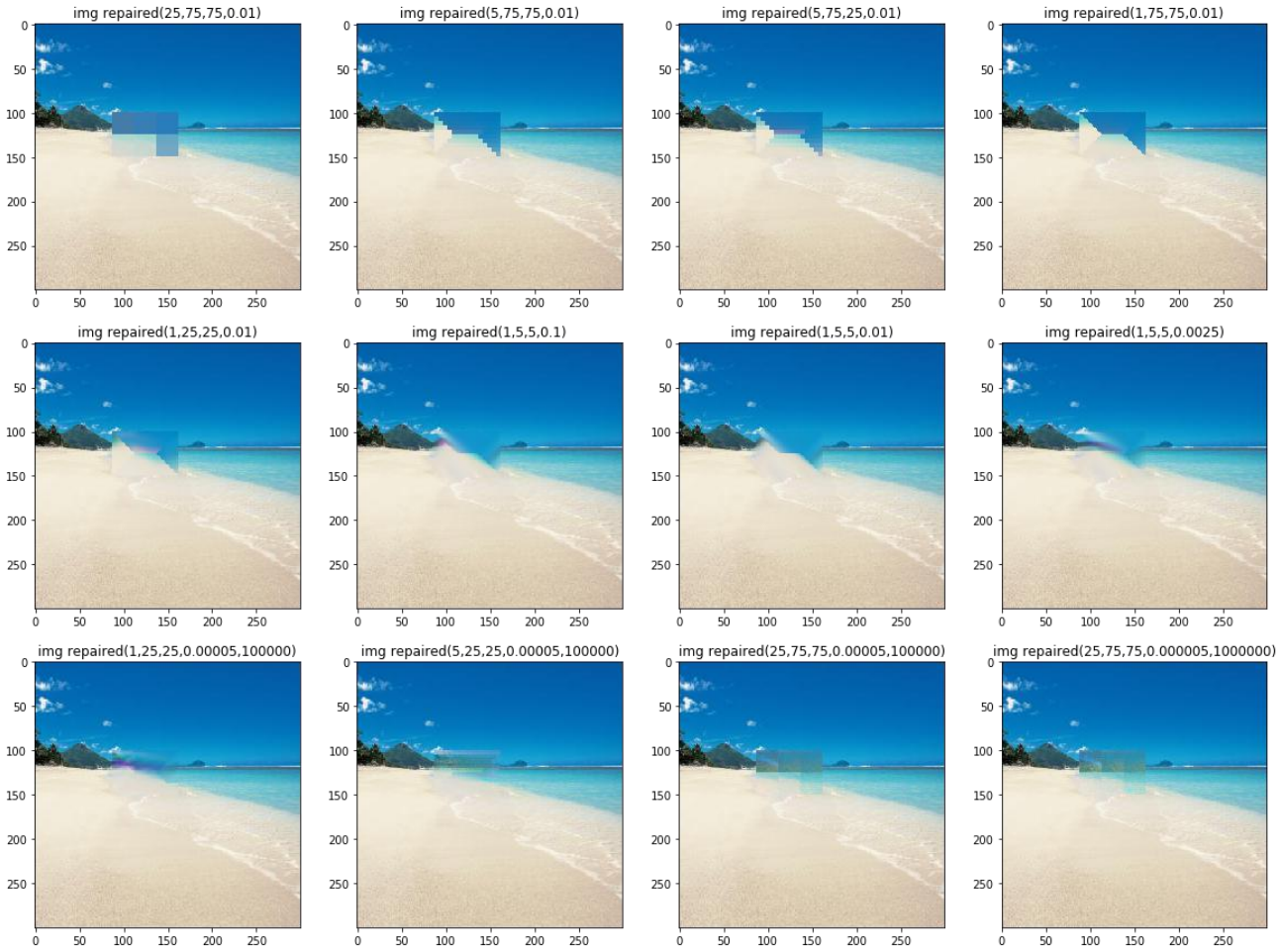
2.2 Réparer des parties manquantes de l'image

2.2.1 Description



La méthode de complétion que nous avons utilisée est illustrée dans la figure ci-dessus. Les parties manquantes sont parcourues en spirale dans le sens des aiguilles d'une montre et les parties avec une longueur de **step** sont complétées séquentiellement (c'est-à-dire que l'ordre de complétion est indiqué dans le numéro de séquence de la figure ci-dessus). Les méthods d'implémenté peuvent se trouvé dans ***mpart_repair.py.***

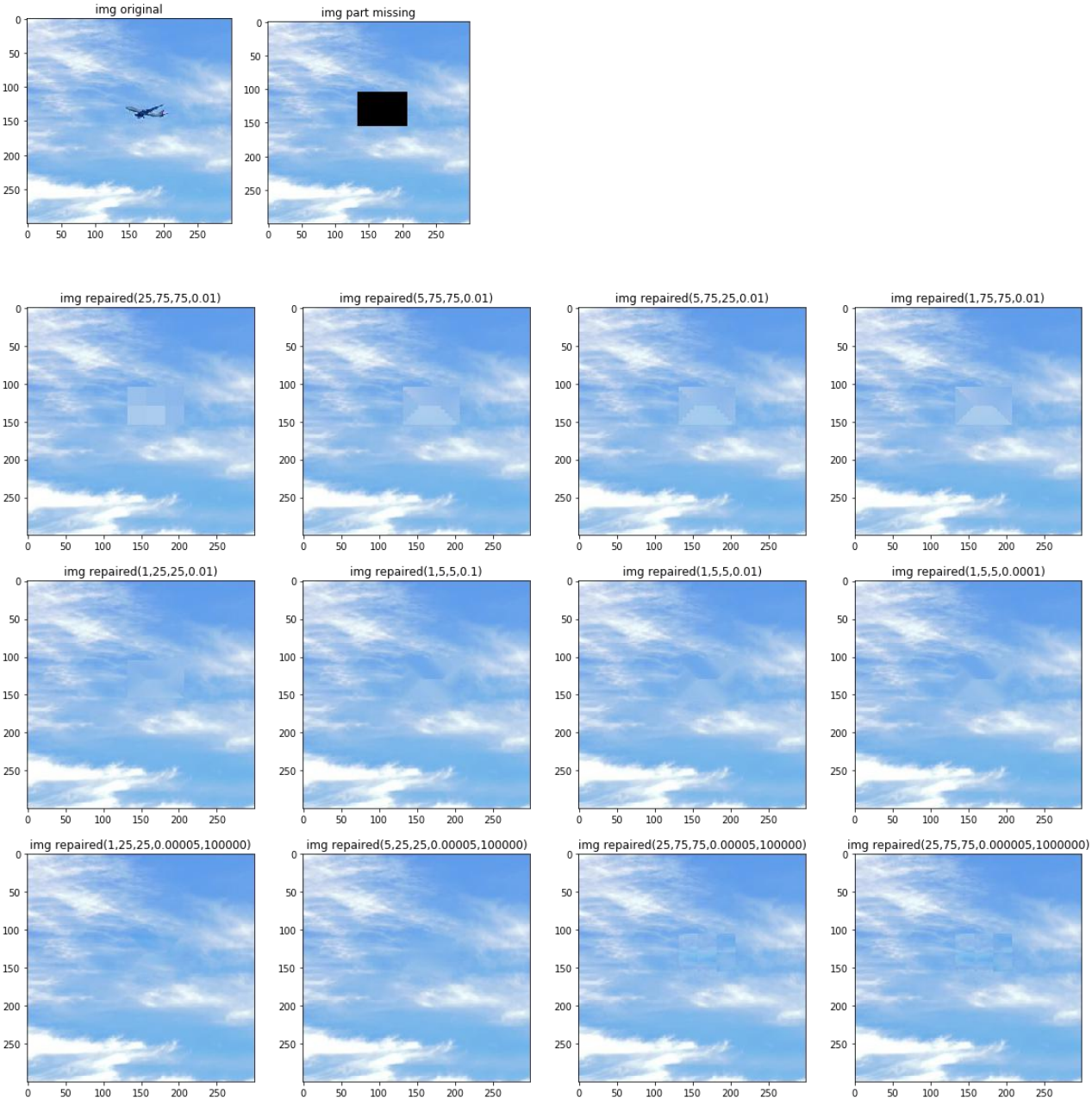
2.2.2 Évaluation des résultats



Les paramètres correspondants entre parenthèses sont: (step,h,d_h=d_w,alpha,max_iter).
Valeur default de max_iter est 1000.
Temps d'exécution (De gauche à droite, de haut en bas):
0.95 s, 25.27 s, 29.88 s, 488.68 s, 102.71 s, 102.71 s, 103.70 s, 134.56 s, 169.58 s, 9.46 s, 0.93 s, 1.09 s

Conclusion: Comme on peut le voir sur les résultats de la figure ci-dessus, réduire la longueur de l'étape est en effet bénéfique pour augmenter la précision de la réparation, mais cela ralentira également considérablement la vitesse de réparation, alors choisissez une longueur d'étape appropriée, en même temps réduisez la valeur de alpha et augmentez le nombre d'itérations, cela c'est plus raisonnable.

Un autre exemple concerne la découpe d'objets dans les images:



2.2.3 l’ordre de remplissage a-t-il une importance ?

L’ordre de remplissage est importante.

Chaque fois que nous effectuons la prochaine réparation sur la base de l'achèvement de la dernière réparation, chaque réparation superposera l'erreur de la dernière prédiction, et l'erreur total devenir de plus en plus importante, ce qui affectera directement les résultats de la réparation Par conséquent, il est nécessaire de réparer d'abord les pixels si le poids est important. Lorsque l'erreur est superposée dans une certaine mesure, les pixels qui sont moins pondérés seront affectés, de sorte que l'impact sur le résultat de la réparation sera relativement faible.