

Travail Module Analyse d'image

2021-2022

Ce travail a porté sur la construction d'un réseau de neurones convolutif sur le langage Python, appliqué à des images astronomiques.

Astronomical Image Reconstruction with Convolutional Neural Networks

Rémi Flamary
Université Côte d'Azur
Lagrange, OCA, CNRS
Nice, France
Email: remi.flamary@unice.fr

Introduction

En astronomie, la qualité de l'image est souvent altérée par des perturbations atmosphériques, causées par des dépressions, l'humidité, ou des Jet-Streams à haute altitude, qui entraînent un étalement de l'image. À cela s'ajoute le pouvoir séparateur de l'optique qui entraînera un étalement d'un point sur une tache plus ou moins grande.

L'objectif de ce travail est d'exposer une méthode basée sur l'intelligence artificielle pour reconstruire des images bruitées, comme présenté dans le papier.

Méthode :

1) Construction du jeu de données

1.a construction images bruitées

Plusieurs images prises par des capteurs du télescope Hubble ont été prises dans une banque de données (https://archive.stsci.edu/cgi-bin/dss_form). Des images de galaxies ont été prises ainsi que des amas globulaires et de nébuleuses pour étoffer la banque d'apprentissage et pouvoir élargir les résultats (11 objets dans ce travail contre 6 dans le papier).

Ces images nettes ont été convolées par un kernel représentant une tache d'Airy pour simuler la Point-Spread-Function. Du bruit gaussien a été ajouté par la suite.

Cela a permis d'obtenir des simulations de diffraction et de bruitage de nos images, comme illustré comme suit.

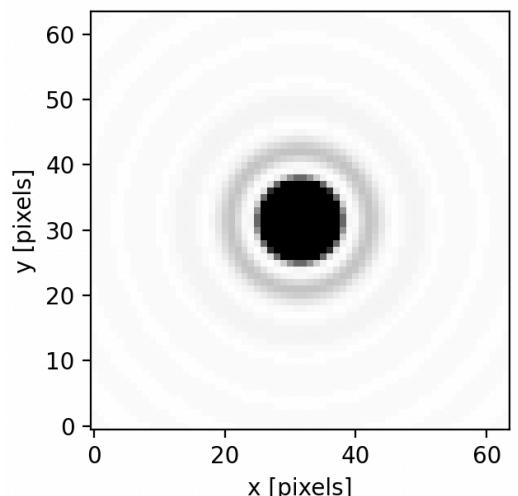


Figure 1 : Kernel de convolution - Python

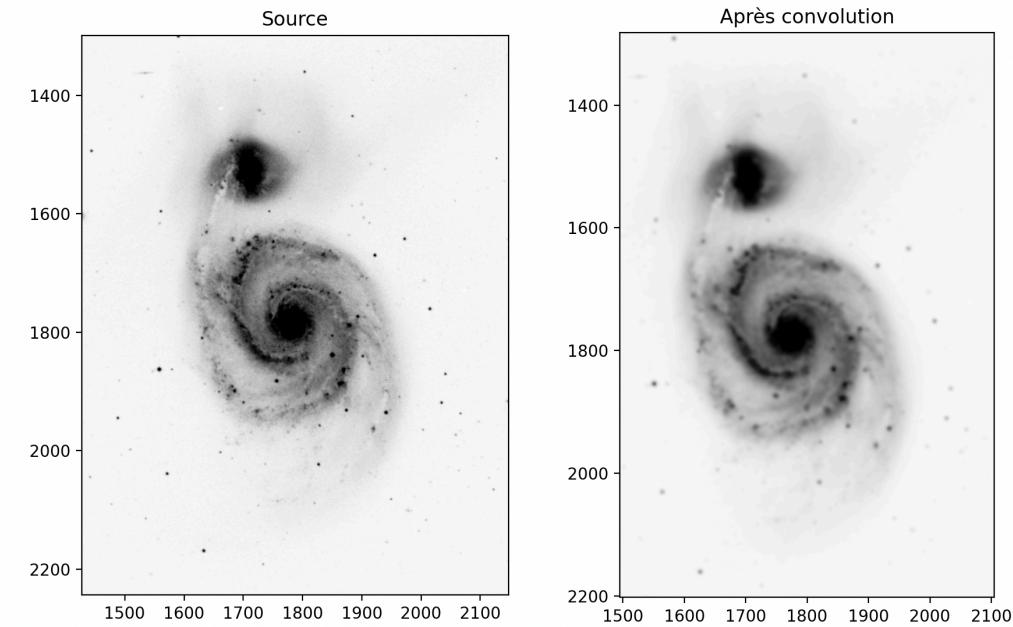


Figure 2 : Image originale (gauche) et bruitée (droite) - Python
Galaxie M51

1.2 Découpage des données

Ces images ont ensuite été aléatoirement découpées sur des zones superposées de 32x32 (comme spécifié dans le papier).

2000 découpes ont été faites par image, correspondant à un volume de 22000 images bruitées et les 22000 images nettes correspondantes. Le tableau suivant en offre quelques exemples.

Cette illustration nous permettent déjà de nous rendre compte d'un des enjeux auquel notre CNN devra faire face : il y a énormément de carrés entièrement noirs, et même sur les carrées contenant des étoiles, l'information « utile » n'occupe que quelques pourcentages de l'image...

| | Sur M31 | Sur M31 | Sur M31 |
|-------------------|---------|---------|---------|
| Avant convolution | | | |
| Après convolution | | | |

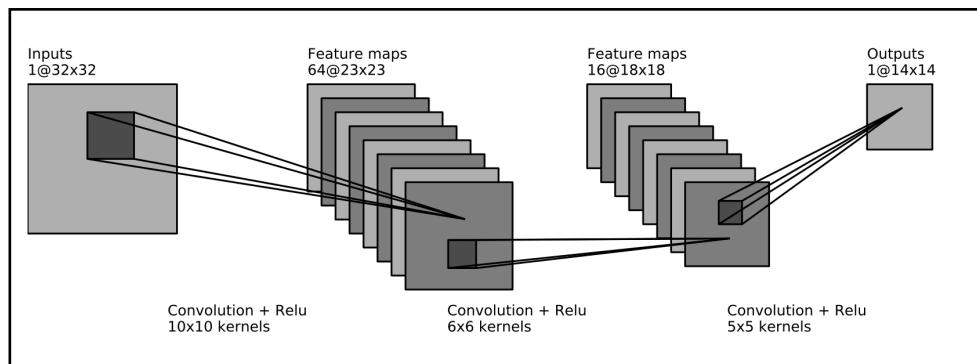
1.3 Répartition des données

Les données ont ensuite été réparties en deux catégories : Train-Set (90%) et Test-Set (10%), pour l'apprentissage et l'évaluation de notre CNN. Elles ont également été normalisées, mélangées aléatoirement.

- Le code source commenté de cette partie correspond aux parties commentées A B C D E F dans le fichier Python
- L'ensemble des images est pré-enregistré dans le répertoire fourni avec le code

2) Construction du CNN

Une fois des fichiers créés, un réseau de neurones convolutif a été construit. L'architecture proposée dans le papier était la suivante, avec 3 couches profondes convolutives, avec une fonction non-linéaire ReLU. Le nombre de kernel est indiqué sur l'image.



2.1 Architecture

Dans le papier, l'auteur n'effectue pas de Padding autour des données, cela signifie que le nombre d'informations diminue au fur et à mesure des couches, puisque la convolution d'un kernel sur l'input qui arrive donne une matrice plus petite.

De ce fait, le CNN donne, pour chaque carré de 32*32 pixels un carré de 14*14 pixels en sortie.

Pour éviter que cette réduction d'information n'affecte la qualité de l'image, et surtout face un problème de comparaison (il aurait fallu réduire la taille de l'image nette à 14x14px par interpolation pour pouvoir la comparer à l'image sortant du CNN...) j'ai pris la liberté d'ajouter à la fin du CNN une couche « Fully-Connected-Layer » avec 1024 (32×32) éléments en sortie. A noter que le papier n'explique pas dans le détail la méthode utilisée pour évaluer ces carrés de 14x14px.

2.2 Fonctions

- Le papier utilisait une fonction d'Optimizer en Stochastic-Gradient-Descent. J'ai néanmoins utilisé un autre optimizer, le **Adam**, n'ayant pas réussi à faire fonctionner le premier de façon satisfaisante avec mes données : le modèle retenait une image et l'affichait systématiquement en sortie avec seulement de légères variations.

- Le Learning rate a été fixé assez bas, à **0.001** avec Adam (contre 0.01 dans l'article avec SGD), à taux d'apprentissage plus élevé le modèle était moins performant.

- La fonction de perte choisie a été la **MSELoss** - Mean-Square-Error. En effet, elle est plus appropriée pour des predictions numériques que d'autres fonctions (CrossEntropy, par exemple). Aucune indication n'était donnée à ce sujet dans l'article.

- Des batch de **50 données** ont été utilisés, comme spécifiés dans le papier. Le dropout des données a également activé.

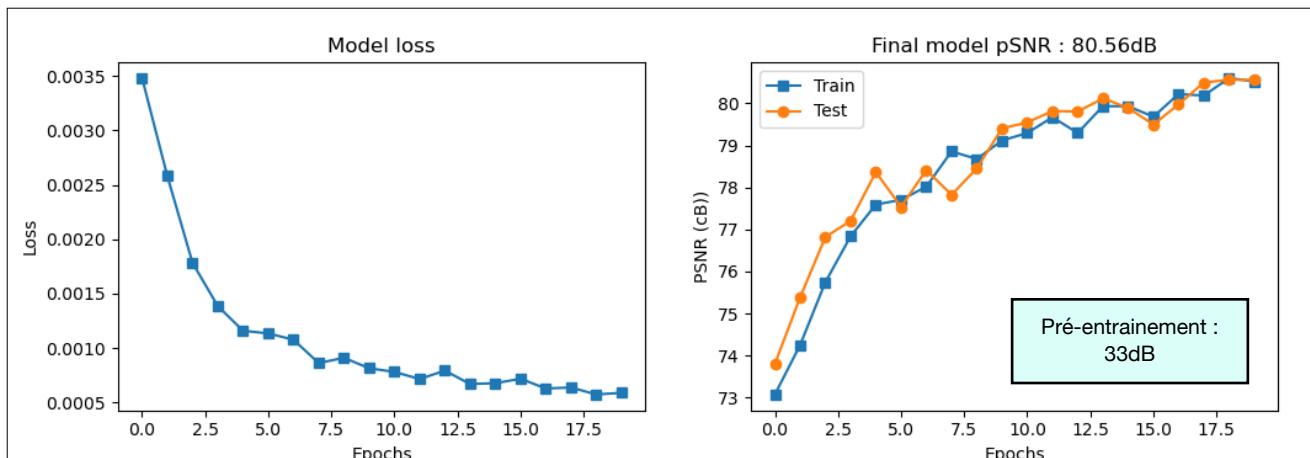
- Du fait du temps de calcul important, seules 20 epochs d'analyse ont été réalisées.

3) Résultats

- En guise d'évaluation, une mesure du Peak-SNR a été faite entre l'image nette et l'image bruitée, ainsi qu'une analyse visuelle.

- Sur notre jeu de données, le pSNR moyen entre l'image Nette et l'image Bruitée est calculé aux alentours de 30 dB. Après passage dans le CNN, voici l'évolution du pSNR au fil des epochs, ainsi que la loss-function.

- Le réseau à 20 epochs peut être sauvegardé et chargé à la fin si besoin.



Notre modèle permet donc bien d'améliorer le pSNR.

4) Limites :

Notre modèle fonctionne mais a quelques limites. Le papier s'est contenté de donner les chiffres du pSNR post traitement (sans donner ceux pré-traitement..). Or, on peut constater visuellement que cette amélioration du pSNR est loin d'être un paramètre suffisant !

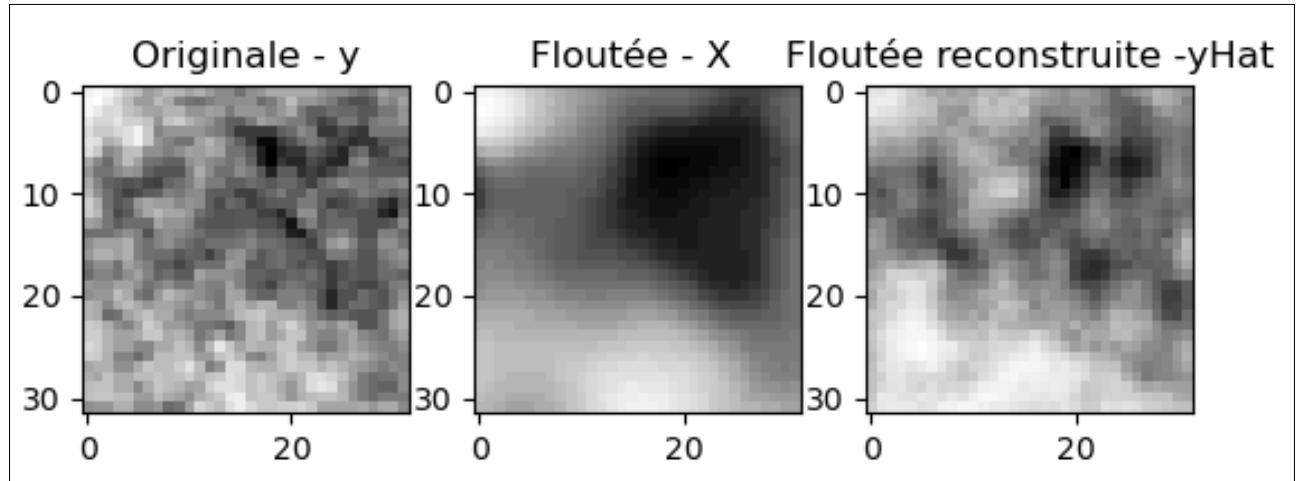
On observe visuellement un effet très marqué de l'apprentissage sur les données, avec un affinement de la correction de dispersion de la tache d'Airy au fil des epochs (ci-dessous images à 20 vs 1 epochs).

Notre CNN fonctionne de façon satisfaisante, néanmoins il faudrait probablement largement augmenter le nombre d'epoch, diversifier les données d'apprentissage, et ajouter des fonctions d'optimisation plus adaptées aux faibles variations dans l'image qu'il doit analyser.

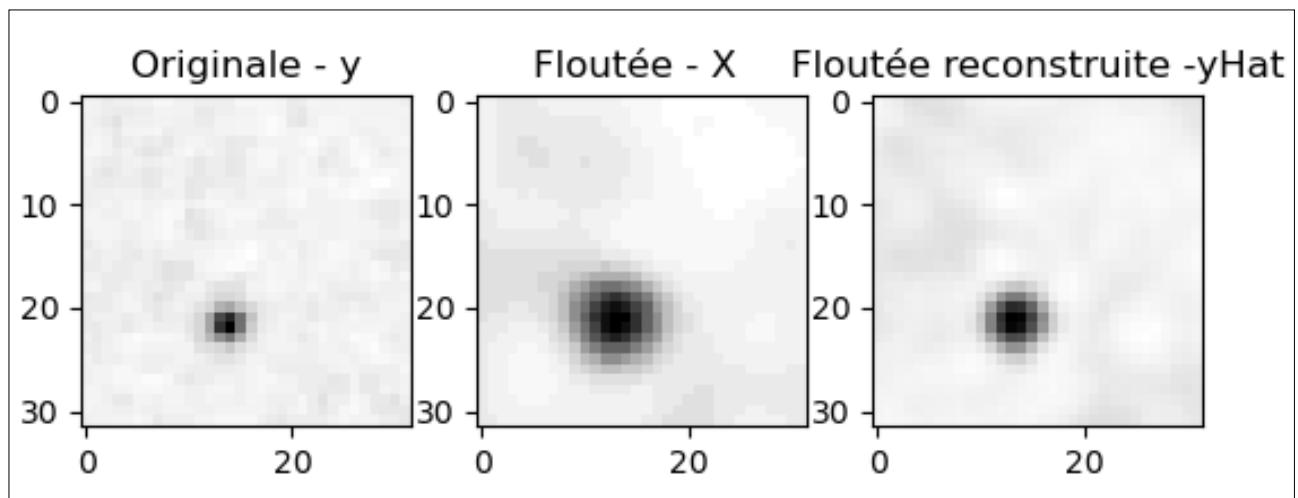
Il semble aussi nécessaire de construire et utiliser un paramètre plus pertinent (sharpness ?) que le pSNR pour évaluer la performance du modèle et la qualité de la reconstruction de l'image, puisque nous voulons surtout reconstruire les étoiles et non le bruit de fond, alors le pSNR analyse l'image dans sa globalité.

- Après entrainement sur 20 epochs :

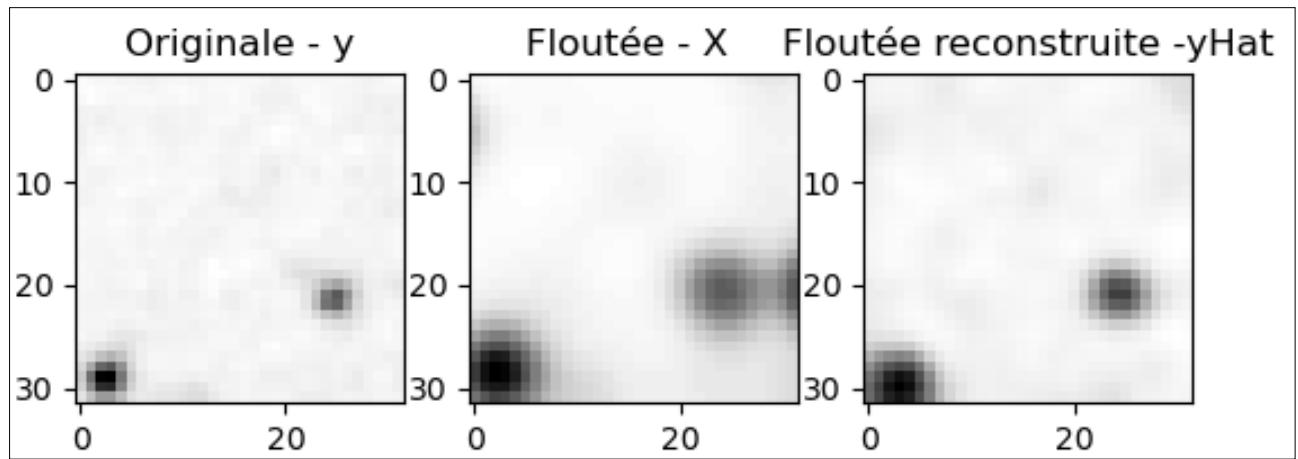
- Amélioration du contraste



- réduction dispersion taille étoile

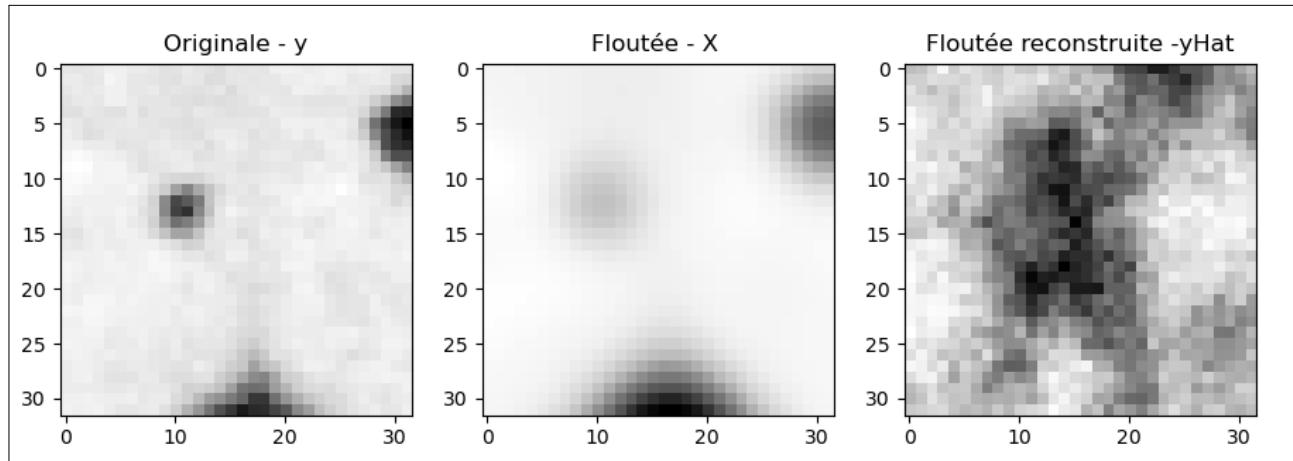
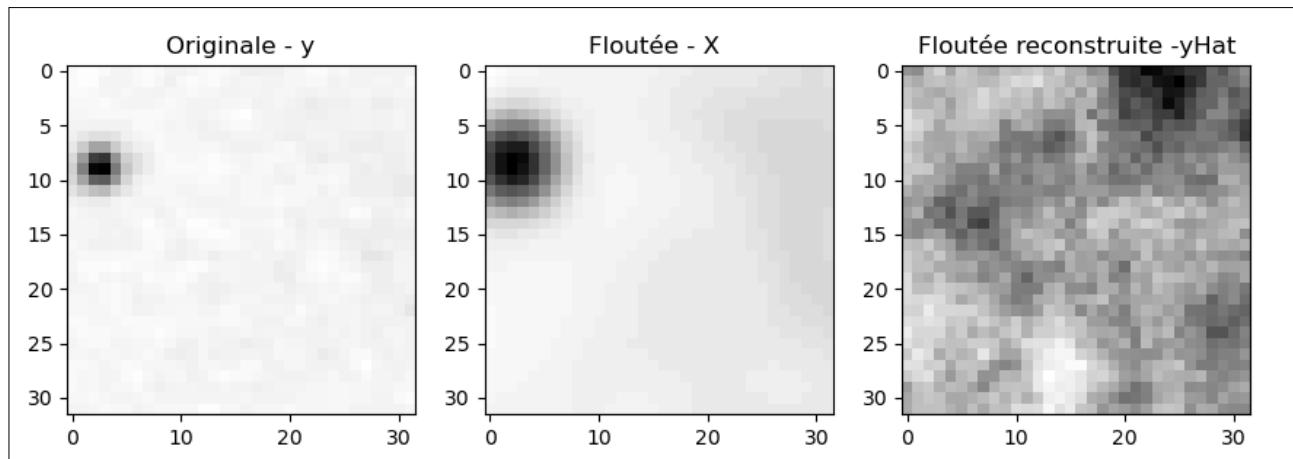
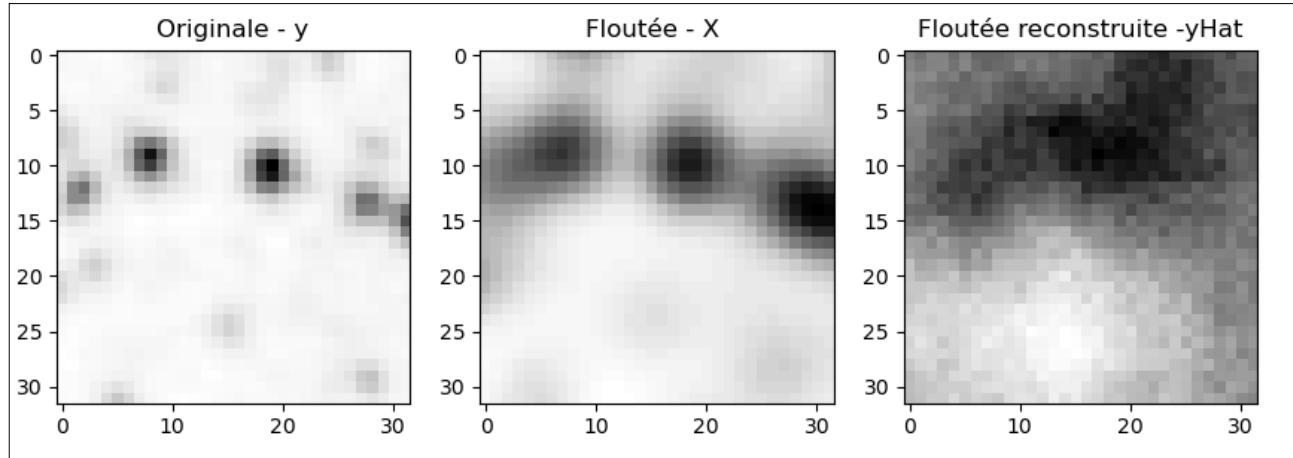


- réduction dispersion taille étoiles + suppression bruit sur la droite



- En comparaison, après entrainement sur 1 epochs :

Reconstruction in-interprétable

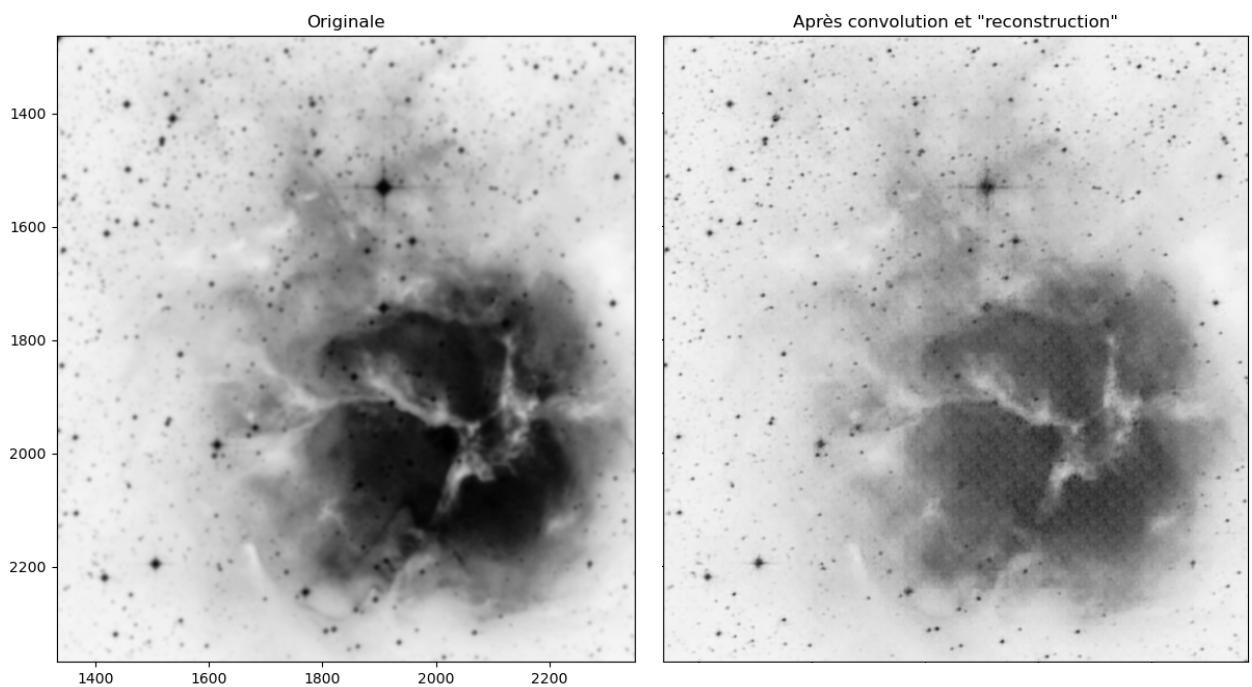
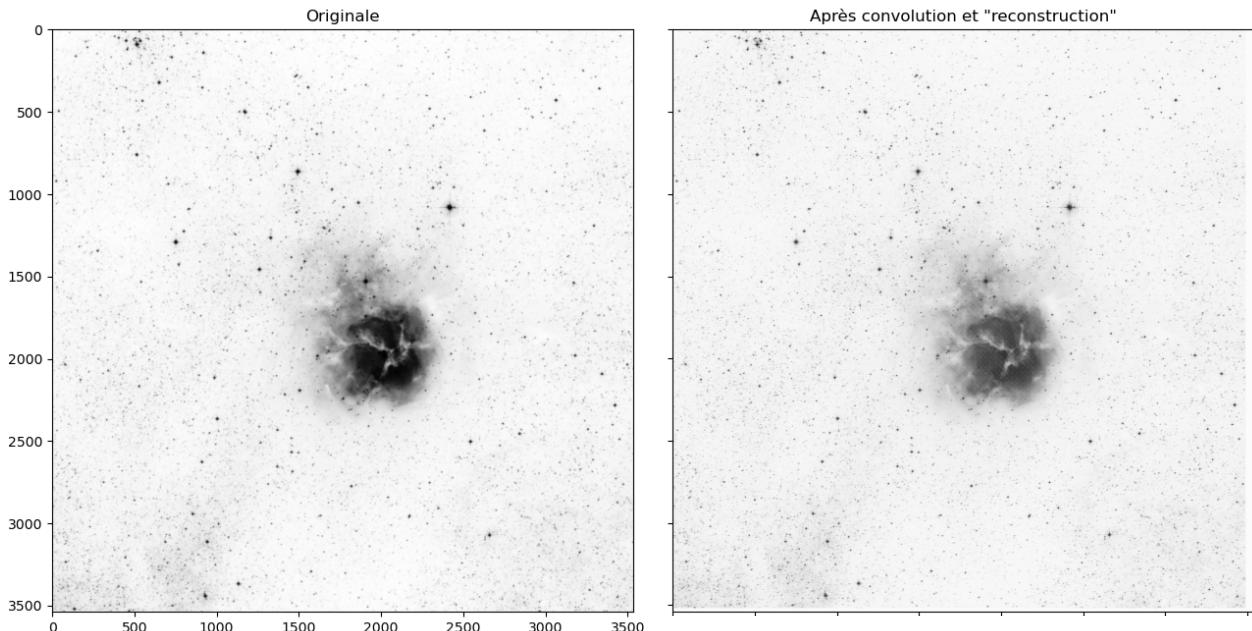


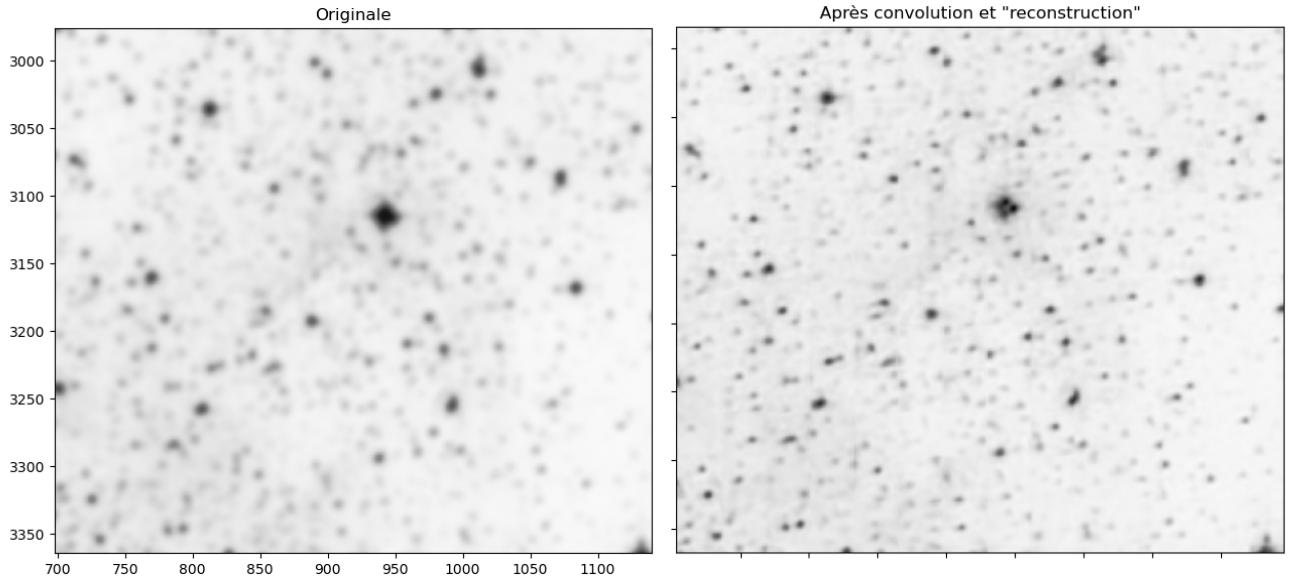
Evaluation sur reconstruction d'une image entière qu'il n'a jamais vue :

- Nébuleuse du trèfle M20 :

L'image est plus nette, mais souffre de l'apposition des carrés de 32*32 côté à côtes lorsqu'on zoom (même si une fenêtre glissante a essayé de corriger de défaut)

Ici « Originale » correspond à l'image bruitée avant passage dans le CNN.





A noter que le modèle fonctionne uniquement sur des images de ciel profond, galaxies, nébuleuses...

Un essai a été fait sur Jupiter, mais s'est avéré moins concluant...

