



# Débruitage et régularisation par réseau de neurones

Projet Industriel - Mai 2019

Book technique réalisé par :

ELKATEB Mohamed  
FAINELLI Faustine  
HOTZEL RUTHER Dominique  
MACHADO SANTOS ROHDE Pedro  
VILLON HUAMAN Alexandra

Encadré par :

Oberlin Thomas  
Soubies Emmanuel

---

À partir de l'article *Deep Image Prior*, D. Ulyanov, A. Vedaldi, V. Lempitsky



## Table des matières

<b>1</b>	<b>Introduction</b>	<b>2</b>
<b>2</b>	<b>Méthode développée dans l'article</b>	<b>3</b>
2.1	Modèle . . . . .	3
2.2	Architecture du réseau . . . . .	4
<b>3</b>	<b>Expériences réalisées</b>	<b>5</b>
3.1	Entrée du réseau . . . . .	5
3.1.1	Entrée du réseau après optimisation . . . . .	5
3.1.2	Entrée du réseau : régularisation durant l'optimisation . . . . .	9
3.1.3	Entrée du réseau avant optimisation . . . . .	9
3.1.4	Optimisation par rapport à l'entrée . . . . .	12
3.2	Profondeur du réseau . . . . .	13
3.3	Skip connections . . . . .	14
3.4	Méthode d'optimisation . . . . .	17
3.5	Robustesse de la méthode . . . . .	20
3.5.1	Performances sur des images contenant du bruit réel . . . . .	20
<b>4</b>	<b>Exécution sur Google Colab</b>	<b>20</b>
<b>5</b>	<b>Conclusion</b>	<b>22</b>

# 1 Introduction

Les réseaux neuronaux convolutifs sont devenus un outil populaire pour la génération et la restauration d'images. Leur performance est en partie liée à leur capacité à apprendre des a priori réalistes à partir d'une base de données d'images.

Toutefois, le présent article montre que la **structure** d'un réseau générateur est suffisante pour obtenir un grand nombre de statistiques d'image, sans phase d'apprentissage préalable. Un réseau neuronal initialisé de façon aléatoire peut être utilisé "à la main" et présente d'excellents résultats en termes de débruitage, super-résolution, inpainting...

Finalement, cette approche permet de rapprocher deux familles très répandues de méthode de restauration d'images :

- les méthodes basées sur *l'apprentissage* utilisant des réseaux convolutifs profonds,
- les méthodes *dépourvues d'apprentissage* basées sur des a priori d'images fabriqués à la main.

Ce book a pour but, premièrement, de rappeler la méthode développée dans l'article et deuxièmement de retracer nos expériences sur le réseau et ses paramètres.

## 2 Méthode développée dans l'article

Au lieu de suivre le paradigme commun de former un réseau convolutif (ConvNet) sur un grand ensemble de données d'images, il convient ici d'ajuster un réseau générateur à une seule image dégradée. Dans ce schéma, les poids de réseau servent à la paramétrisation de l'image restaurée. Ces poids sont initialisés et ajustés de manière aléatoire afin de **maximiser leur probabilité** en fonction d'une image dégradée spécifique et d'un modèle d'observation dépendant de la tâche. En d'autres termes, il faut considérer la reconstruction comme un **problème de génération** d'image conditionnelle. La seule information nécessaire pour le résoudre est contenue dans l'image d'entrée unique, dégradée et la structure du réseau lui-même utilisée pour la reconstruction.

### 2.1 Modèle

Un ConvNet peut être vu comme une fonction paramétrique  $x = f_\theta(z)$ , qui "mappe" un vecteur  $z$  sur une image  $x$ . Habituellement, on s'attendrait à ce que les paramètres du réseau soient appris à partir des données pour que le réseau acquiert des informations utiles sur les images. Au lieu de cela, on choisit d'interpréter le réseau de neurones comme une paramétrisation  $x = f_\theta(z)$  de l'image  $x \in \mathbb{R}^{3 \times H \times W}$ . Le vecteur  $z$  est en fait un tenseur aléatoire fixe tel que :  $z \in \mathbb{R}^{C \times H \times W}$  et le réseau mappe les paramètres  $\theta$ , contenant les poids et le biais des filtres dans le réseau, à l'image  $x$ . Le réseau a une structure classique et présente une alternance entre opérations de filtrage telles que des fonctions de convolution linéaire, de suréchantillonnage et d'activation non-linéaire. Sans entraînement, on ne peut pas attendre du réseau  $f_\theta$  qu'il connaisse des concepts de "haut niveau" tels que l'apparition de visages humains ou d'animaux. Cependant, il est montré que le réseau contient des connaissances sur la **structure de "bas niveau"** des images naturelles. Cette connaissance est suffisante pour modéliser la *distribution d'image conditionnelle*  $p(x|x_0)$  où l'image  $x$  est déterminée à partir d'une observation corrompue  $x_0$ . Plutôt que de travailler avec des distributions, le problème est posé comme une minimisation d'énergie :  $x^* = \min_x E(x; x_0) + R(x)$  avec  $E(x; x_0)$  un terme de données dépendant de l'application,  $x_0$  une image bruyante et  $R(x)$  un régularisateur. Un exemple simple de régularisateur est par exemple la variation totale (total variation en anglais), qui encourage les images à contenir des régions uniformes. Ici, à la place du régularisateur explicite  $R(x)$ , un a priori implicite est utilisé, celui-ci ayant été capturé par la paramétrisation du réseau neuronal, comme suit :  $\theta^* = \operatorname{argmin}_\theta(f_\theta(z); x_0)$ ,  $x^* = f_{\theta^*}(z)$ . Le minimiseur (local)  $\theta^*$  est obtenu à l'aide d'un optimiseur : la descente de gradient. Celui-ci est obtenu en partant d'une initialisation aléatoire des paramètres  $\theta$ , de sorte que la seule information disponible soit l'image bruyante  $x_0$ . Le résultat du processus de restauration est alors donné sous la forme  $x^* = f_{\theta^*}(z)$ . Étant donné qu'aucun aspect du réseau  $f_\theta$  n'est appris à partir de données à l'avance, on peut dire qu'une telle image est effectivement réalisée "à la main".

## 2.2 Architecture du réseau

Le choix de l'architecture a un impact sur les résultats. La structure retenue ici est une architecture de **type «hourglass»** (sablier) ou bien décodeur-encodeur, avec des skip connections (connexions de saut), où  $z$  et  $x$  ont les mêmes dimensions spatiales et où le réseau compte plusieurs millions de paramètres :

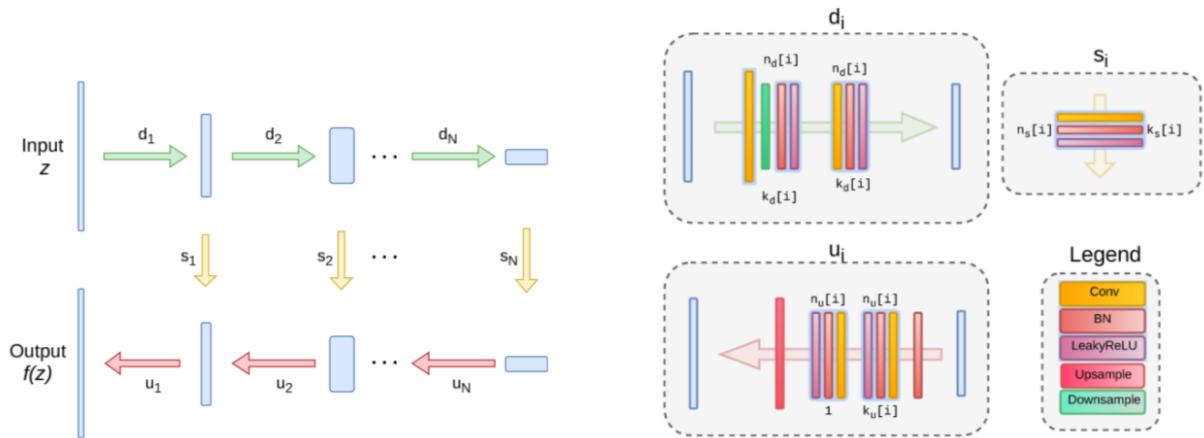


FIGURE 1 – Architecture utilisée dans les expériences. Nous ajoutons parfois des skip connections (flèches jaunes).  $n_u[i]$ ,  $n_d[i]$ ,  $n_s[i]$  correspondent respectivement au nombre de filtres sur la couche de profondeur  $i$  pour les connexions de suréchantillonnage ( $n_u$ ), de sous-échantillonnage ( $n_d$ ) et de saut ( $n_s$ ). Les valeurs  $k_u[i]$ ,  $k_d[i]$ ,  $k_s[i]$  correspondent aux tailles des noyaux respectifs.

L'encodeur crée une représentation succincte et compacte de l'image, souvent appelée "mot code" ou "représentation de l'espace latent". Le décodeur reconstruit l'image à partir de ces "représentations latentes". Souvent, les poids du décodeur sont juste une transposition de ceux de l'encodeur, les décodeurs et les encodeurs sont donc **symétriques** et partage les mêmes poids.

À l'échelle du code, le réseau est implémenté par la fonction `get_net.py`

```

elif fname == 'data/denoising/F16_GT.png':
    num_iter = 3000
    input_depth = 32
    figsize = 4

    net = get_net(input_depth, 'skip', pad,
                  skip_n33d=128,
                  skip_n33u=128,
                  skip_n11=4,
                  num_scales=5,
                  upsample_mode='bilinear').type(dtype)

```

FIGURE 2 – La variable `num_iter` définit le nombre d’itérations lors de la phase d’optimisation, `input_depth` correspond à la profondeur du réseau neuronal : ici nous travaillons avec un réseau de 32 couches. `skip_n11` représente le nombre de skip connections, `skip_33d` et `skip_33u` représentent respectivement le nombre de canaux dans les couches montantes (upsample) et descendentes (downsample).

## 3 Expériences réalisées

### 3.1 Entrée du réseau

#### 3.1.1 Entrée du réseau après optimisation

On a d’abord essayé de jouer avec leur réseau tel qu’il était. Après optimisation du réseau pour approcher une image donnée, on a joué sur l’entrée  $Z$  de ce réseau. On a fait quelques expériences, telles que : ajouter des constantes à  $Z$  ( $Z + k$ ), multiplier  $Z$  par des constantes ( $kZ$ ), ajouter du bruit à  $Z$  ( $Z + b, b \sim N(0, \frac{1}{30})$ ) et on a aussi mis des  $Z$  différents mais de même loi que l’original ( $Z \sim U(0, 0.1)$ ).

On remarque que l’ajout d’un bruit en entrée (figure 4) produit des artefacts dans l’image générée. Cela peut indiquer qu’une optimisation sur l’entrée après l’optimisation sur le réseau peut encore améliorer la performance du débruitage. En effet, dans leur code original, ils ajoutent un bruit en entrée à chaque itération : *During fitting of the networks we often use a noise-based regularization. I.e. at each iteration we perturb the input  $z$  with an additive normal noise with zero mean and standard deviation  $\sigma_p$ . While we have found such regularization to impede optimization process, we also observed that the network was able to eventually optimize its objective to zero no matter the variance of the additive noise (i.e. the network was always able to adapt to any reasonable variance for sufficiently large number of optimization steps).*

On trouve que, en multipliant tous les coefficients par la même constante positive  $k > 0$  (figure 5), on obtient la même image en sortie, pour n’importe quelle constante.

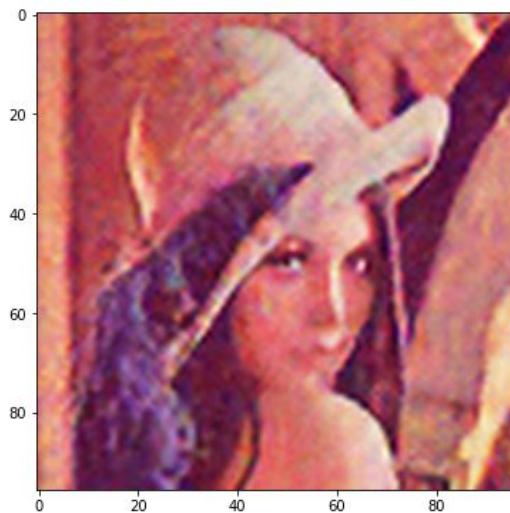


FIGURE 3 – Sortie du réseau avec une entrée Z donnée

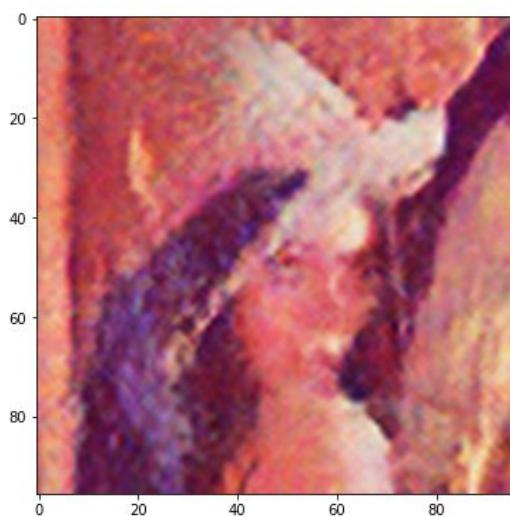


FIGURE 4 – Sortie du réseau avec l'entrée Z perturbée par un bruit additif

$$f_\theta(z) = f_\theta(kz), \forall k > 0$$

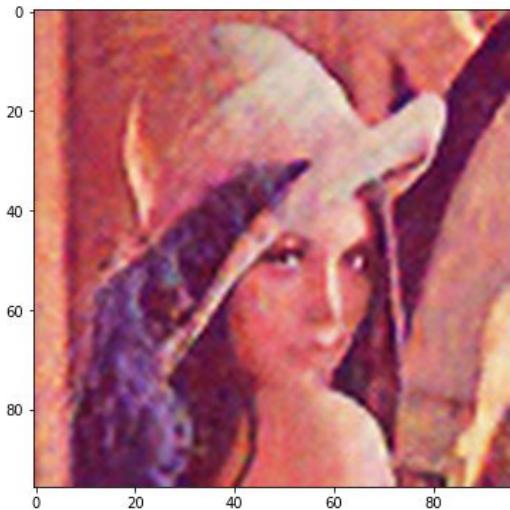


FIGURE 5 – Sortie du réseau avec l'entrée Z multipliée par une constante positive

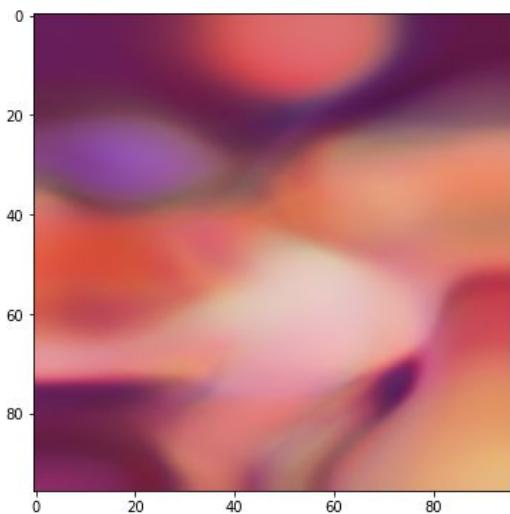


FIGURE 6 – Sortie du réseau avec l'entrée Z multipliée par zéro

En multipliant les coefficients en entrée par zéro (c'est-à-dire, en les prenant tous nuls), on obtient une image en sortie (figure 6) qui a les mêmes couleurs de la sortie originale, mais avec juste des régions à peu près constantes. On ne garde pas les formes de l'image.

En multipliant par une constante négative (figure 7), contrairement à la constante positive, on n'obtient pas du tout la même image. L'image générée ici garde les mêmes couleurs et un peu de texture, mais on n'arrive pas à reproduire les formes. Il faut noter aussi qu'on obtient ce même résultat pour n'importe quelle constante négative.

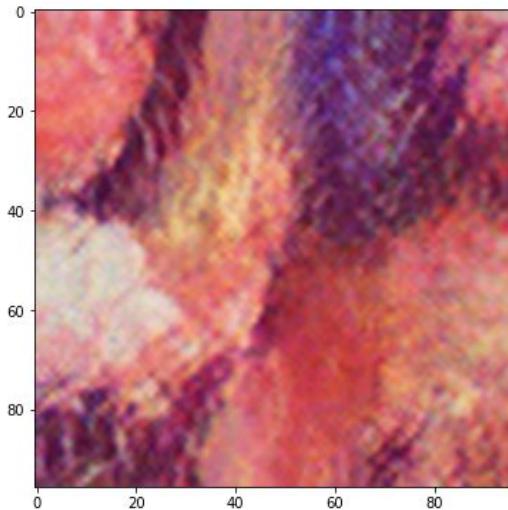


FIGURE 7 – Sortie du réseau avec l'entrée Z multipliée par une constante négative

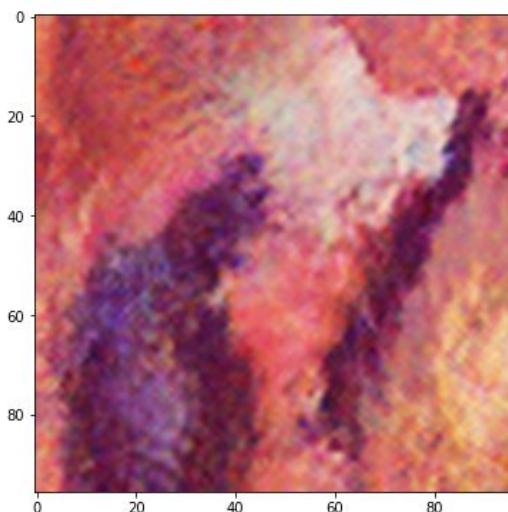


FIGURE 8 – Sortie du réseau avec une entrée Z de même loi de l'originale

Enfin, on prend une entrée de même loi que l'entrée originale, soit  $Z \sim U(0, 0.1)$ . Comme pour la constante négative, on garde les couleurs et on produit des textures, mais les formes ne sont pas du tout les mêmes. L'image générée change à chaque fois qu'on prend un autre tenseur aléatoire.

Par ailleurs, on a également fait des expériences avec une constante additive, mais on a trouvé que l'image générée restait inchangée pour n'importe quelle constante, positive ou négative.

### 3.1.2 Entrée du réseau : régularisation durant l'optimisation

Comme on l'a vu précédemment, on fait une régularisation de l'entrée en y ajoutant du bruit à chaque itération ( $Z' = Z + b$ ,  $b \sim N(0, \frac{1}{30})$ ). Il se trouve que, du fait que cette régularisation ralentisse l'optimisation, le plateau de décision pour prendre la meilleure image générée (avant que le réseau ne commence à modéliser aussi le bruit) devient plus large, comme montré dans la figure 9.

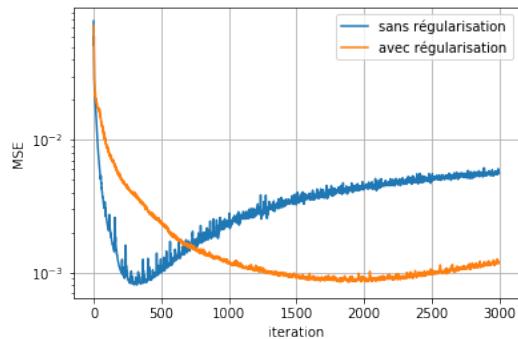


FIGURE 9 – MSE entre image générée et bruitée pour régularisations de  $\sigma = 0$  et  $\sigma = \frac{1}{30}$

### 3.1.3 Entrée du réseau avant optimisation

Une autre façon de tester cet algorithme a aussi été de jouer sur l'entrée du réseau avant son optimisation. On a pris un  $Z$  avec beaucoup (90%) de coefficients nuls (figure 11) et on a optimisé le réseau pour approcher la même image et comparer les temps de convergence.

On remarque des courbes des figures 12 et 13 que la convergence est moins rapide, mais on y arrive quand même. Cela nous indique que c'est possible d'utiliser ce réseau avec des entrées de taille plus petite et peut-être aussi avec un réseau plus petit (moins de couches etc.).

#### *Impact de la dimension spatiale de l'entrée sur la convergence :*

Le réseau tel que présenté par l'article référence, part d'une entrée  $Z$  de dimension 32 : on a appelé dimension le nombre de canaux de  $Z$  puisqu'on garde dans ce qui suit  $L$  et  $W$  de l'entrée égaux à ceux de l'image à débruiter. Ce choix bien que fonctionnel est peu justifié par les auteurs. Le résultat obtenu pour  $Z$  parcimonieux, laisse suggérer qu'une réduction de la complexité algorithmique du réseau est envisageable en réduisant la dimension de  $Z$ . Nous avons ainsi observé l'allure de la convergence du MSE et le résultat de débruitage sur une même image pour entrée de dimension supérieure à la dimension de l'image à débruiter, égale à l'image et inférieure. Les résultats de ces tests sont présentés sur la figure 14.

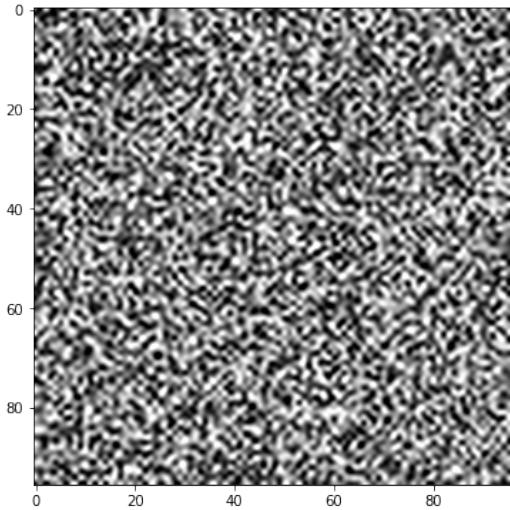


FIGURE 10 – Un des 32 canaux en entrée normale (non-sparse)

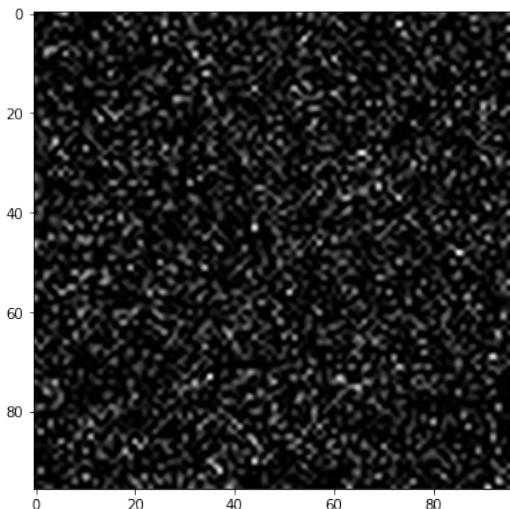


FIGURE 11 – Un des 32 canaux en entrée sparse

Il est possible de dire en observant les courbes précédentes que la dimension de  $Z$  n'a pas de véritable effet sur la convergence du réseau. La courbe de MSE n'étant pas un critère fiable pour confirmer ce constat, nous avons eu recours à une estimation subjective de la qualité d'image résultante. Nous pouvons affirmer après observation des résultats que la qualité est sensiblement conservée après réduction de la dimension de  $Z$ . Ceci valide l'hypothèse que pour l'opération de débruitage, on peut réduire la complexité de l'algorithme sans impacter les performances finales en diminuant la dimension de l'entrée.

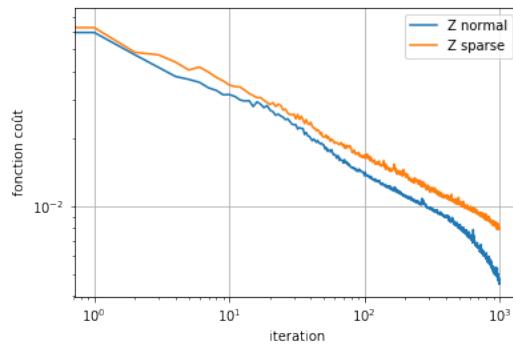


FIGURE 12 – Fonction coût avec Z sparse et non-sparse

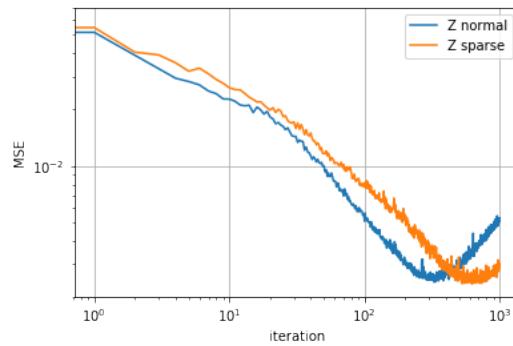


FIGURE 13 – MSE entre image générée et image non bruitée pour un Z sparse et non-sparse

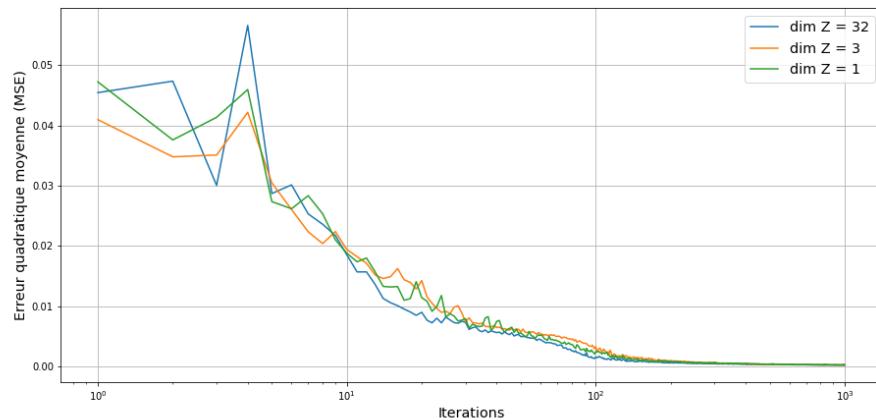


FIGURE 14 – Evolution de la fonction coût (MSE) pour 3 dimensions de l'entrée Z

### 3.1.4 Optimisation par rapport à l'entrée

L'entrée du réseau de neurones étant un tenseur aléatoire, et par la suite indépendant de l'image à débruiter, on peut se permettre de supposer qu'une optimisation par descente de gradient faite sur ce degré de liberté additionnel permettrait d'obtenir un résultat similaire à ce que donnerait une optimisation sur les paramètres de l'architecture neuronale.

En initialisant les paramètres du réseau à des valeurs aléatoires figées, nous avons effectué une optimisation par rapport à l'entrée, en gardant les paramètres  $\theta$  du réseau inchangés (aléatoirement initialisés). Pour cette expérience, nous avons adopté les conditions décrites dans l'article pour le débruitage, en particulier 3000 itérations d'optimisation.

Comme le montre la figure 15, même si une évolution est visible, l'algorithme semble ne pas pouvoir converger, ou alors qu'une possible convergence nécessiterait un nombre d'itérations énorme.

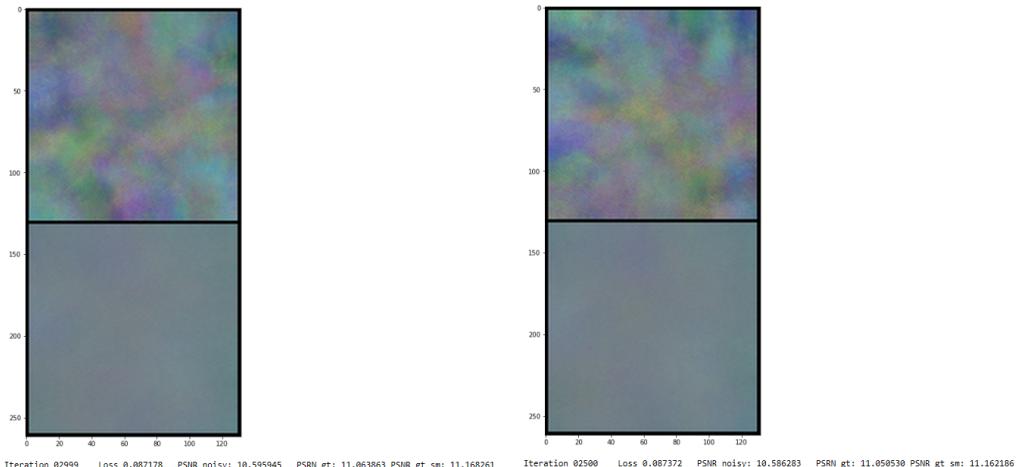


FIGURE 15 – Sortie du réseau à l'itération 2500 et 2999 pour une optimisation sur l'entrée Z

En raison des limitations de la puissance de calcul, nous n'avons pas augmenté davantage le nombre d'itérations. C'est pour cela nous n'avons pas pu affirmer ou rejeter l'hypothèse que l'optimisation par rapport à l'entrée mène à une convergence. Par contre, nous avons essayé de voir, à la lumière des transformations appliquées sur Z après convergence de l'optimisation sur les paramètres du réseau, si une optimisation sur Z affinerait-elle la qualité du débruitage. Pour ce test, on arrête l'optimisation sur le réseau après 1000 itérations suivie de 1000 itérations d'optimisation sur l'entrée Z ayant un nombre de canaux égal à 32 . La figure 16 montre le résultat de cette expérience.

Par la figure 16, il apparait que l'optimisation sur Z, laisse quasiment inchangée la MSE minimale à laquelle le réseau a convergé. Pour une meilleure visualisation nous avons

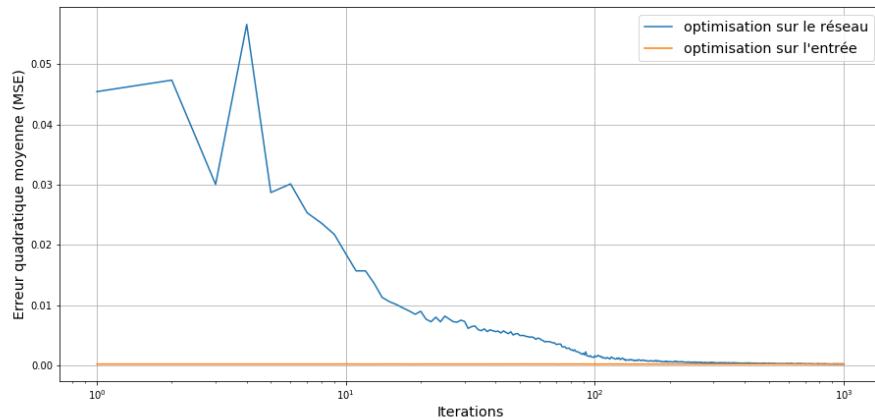


FIGURE 16 – Evolution de la MSE pour une optimisation sur le réseau suivie d'une optimisation sur l'entrée

extrait la courbe d'optimisation sur Z sur la figure 17. Cette dernière confirme qu'il ne s'agit pas d'une courbe de convergence. Par la suite l'optimisation des paramètres du réseau a fait de sorte que le Z initial soit l'optimal pour le MSE final.

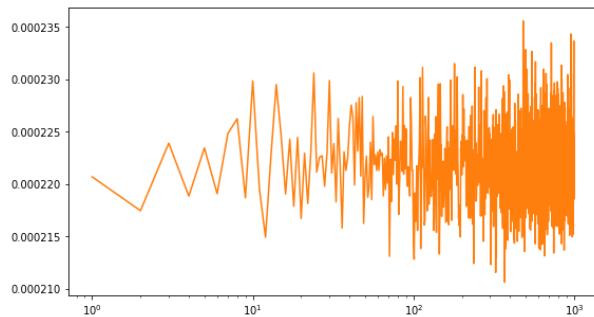


FIGURE 17 – Détails de l'optimisation sur l'entrée (Zoom sur la figure 16)

### 3.2 Profondeur du réseau

La profondeur du réseau de neurone est un paramètre important qui d'après la littérature reste empirique. Néanmoins, plusieurs travaux affirment qu'un réseau profond a tendance à dépasser en performance des réseaux ayant un nombre de couches plus faible. Ceci est expliqué par le fait qu'empiler les couches sert à l'algorithme pour "compresser" davantage les données issues de l'entrée et ainsi obtenir une meilleure représentation de l'entrée.

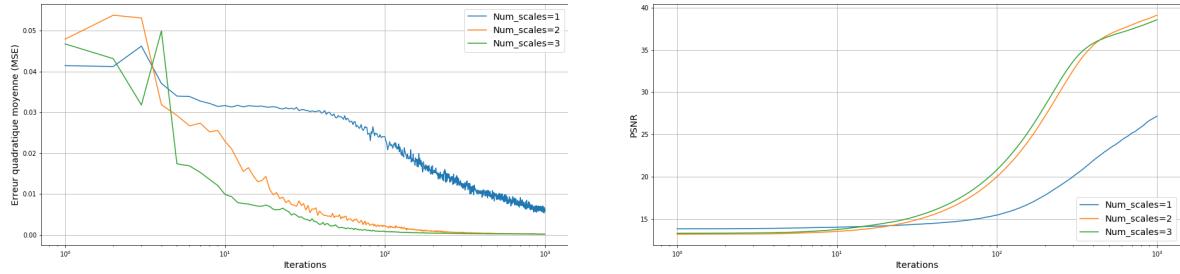


FIGURE 18 – Evolution du MSE et du PSNR pour 3 profondeurs du réseau

La figure 18 montre les résultats obtenus en terme de fonction coût du réseau et de PSNR, en changeant pour une image donnée le nombre de couches convolutives. On remarque que la fonction de coût descend plus rapidement plus le réseau est profond et que le PSNR est sensiblement amélioré. Nous avons également pu constaté que même si la convergence du MSE s'améliore à mesure qu'on empile les couches, le PSNR tend à avoir la même allure à partir d'un certain nombre de couches(ici 2). Il reste à vérifier que ce résultat est bien une caractéristique indépendante d'autres paramètres comme la taille de l'image par exemple.

### 3.3 Skip connections

Une des choses que nous nous sommes proposés de faire dans ce projet était de tester les limites du ConvNet. En analysant l'architecture du réseau, nous avons choisi d'examiner le paramètre *skip\_n11*. Pour cela, nous l'avons fait varier aux extrêmes et comparer aux résultats obtenus avec le réseau d'origine, tel que présenté par les auteurs.

Lorsque nous avons changé ces valeurs, l'un des estimateurs que nous avons examiné est l'erreur quadratique moyenne (en anglais *mean squared error* ou MSE), définie par l'équation 1 et qui mesure la distance entre la sortie et l'image originale, de référence, afin d'avoir une mesure du fonctionnement du ConvNet pour le débruitage de l'image.

$$MSE = E_{\hat{\theta}} \left[ (\hat{\theta} - \theta)^2 \right] \quad (1)$$

Nous avons modifié le paramètre *skip\_n11*, qui valait quatre à l'origine, aux extrêmes zéro et 10 pour vraiment voir l'impact qu'il a sur les résultats, et nous les avons tracés : voir figure 19.

Les courbes obtenues démontrent ce qui est attendu : la MSE décroît à mesure que le nombre d'itérations augmente, mais atteint ensuite un point où le réseau commence à bien approcher l'erreur ainsi que l'image ; il commence alors à grandir. Ce qui s'est passé lorsque

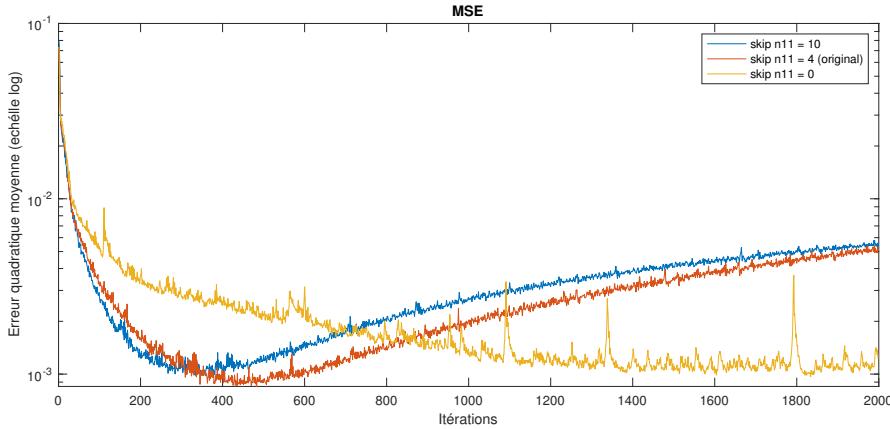


FIGURE 19 – Erreur Quadratique Moyenne

nous avons fait varier le nombre de couches  $skip\_n11$ , c'est que la durée de ce minimum dans la MSE a également varié.

Lorsque nous avons mis une grande quantité de couches ( $skip\_n11 = 10$ ), l'erreur minimale a été trouvée quelques itérations avant et il a remonté peu d'itérations après. En outre, la valeur du minimum était supérieure à celle trouvée avec l'architecture d'origine, indiquée en rouge dans la figure 19.

Une fois que nous avons retiré toutes ces couches ( $skip\_n11 = 0$ ), l'erreur a pris beaucoup plus d'itérations pour atteindre son minimum, mais elle a été conservée plus longtemps. Étant donné qu'il s'agit d'une tâche qui nécessite une certaine réflexion pour choisir l'image avec le moins grand nombre de MSE, il peut s'agir d'une caractéristique intéressante ayant une plage de choix plus étendue. Il a des applications où on n'est pas exactement sûr sur cela qu'on cherche comme sortie, donc avoir une plus grande plage où l'erreur est minimale peut être souhaité. On remarque que la valeur minimale du MSE trouvée est supérieure à la valeur trouvée avec l'architecture originale.

Un tableau récapitulatif est donné ci-dessous :

Nombre de couches $skip\_n11$	MSE
Zéro	9.6463e-04
Quatre (réseau originale)	8.5043e-04
10	9.7528e-04

Un autre estimateur que nous avons ensuite examiné est la fonction de coût, qui mesure l'erreur entre la sortie du réseau et l'image bruyante.

Une des choses soulignées par les auteurs est que le réseau a ce qu'ils appellent une « impédance élevée au bruit », c'est-à-dire que l'erreur ne sera pas bien approximée par le

réseau jusqu'à un certain point. Cela est montré dans l'image 20 extraite de l'article des auteurs.

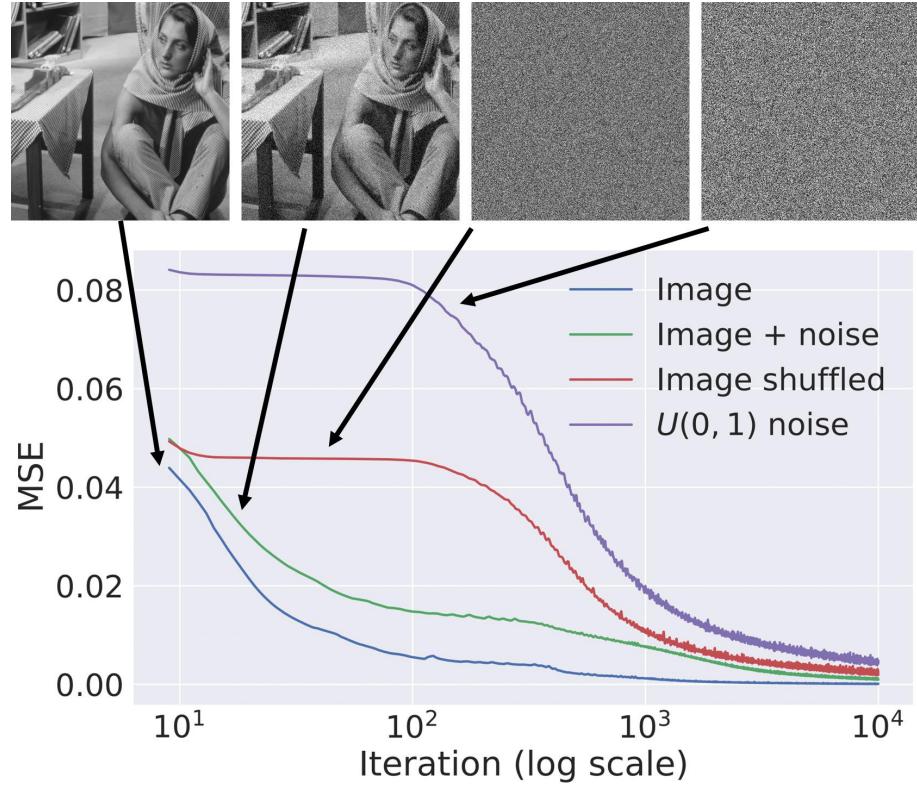


FIGURE 20 – Courbes d'apprentissage pour la tâche de reconstruction en utilisant : une image naturelle, la même image plus du bruit i.i.d., la même avec les pixels mélangés aléatoirement et du bruit blanc.

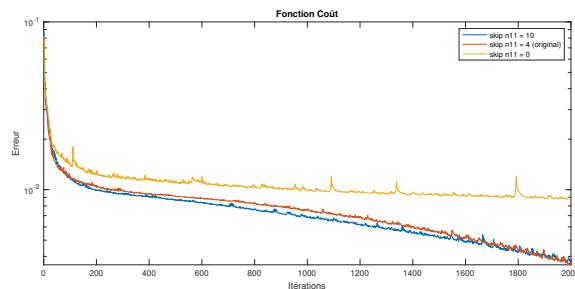


FIGURE 21 – Fonction Coût

Cette même caractéristique est mise en évidence dans le graphique de la figure 21, où la fonction coût va éventuellement à zéro, indiquant que la sortie du réseau est finalement la même que l'image bruyante. Dans le cas où nous avons supprimé les skip connections, cela ne se produit pas dans les 2000 premières itérations (cela peut arriver après un grand

nombre d'itérations, comme dans Figure 20. Cela montre que la modification de l'architecture du réseau permet de moins d'*overfitting*.

### 3.4 Méthode d'optimisation

Le choix de l'architecture du réseau a un grand effet sur la façon dont on cherchera à résoudre le problème de reconstruction.

Dans un premier temps, on a fait des expériences avec une méthode d'optimisation de type descente de gradient laquelle **adapte le taux d'apprentissage de chaque paramètre du réseau** et a la particularité d'estimer de "moments" ( $m_t, v_t$ ) où  $m_t$  représente le moment de premier ordre du gradient(moyenne) et  $v_t$  représente le moment de second ordre (variance). Ci-dessous l'algorithme **ADAM** pas à pas.

Soient les taux de dégradations :  $\omega_1$  et  $\omega_2$ ,  $\sigma$  :une petite constante de stabilisation numérique et  $\theta$  les paramètres initiaux

- Initialisation :  $m_t = 0$  et  $v_t = 0$
- Tant que t<itérations faire
- Calculer le gradient :  $\mathbf{g} = \nabla_{\theta} \Sigma E(f_{\theta}(z), x_0)$
- $t=t+1$
- Mise à jour des moments :  $\mathbf{m} = \omega_1 * \mathbf{m} + (1 - \omega_1) * \mathbf{g}$ ;  $\mathbf{v} = \omega_2 * \mathbf{v} + (1 - \omega_2) \mathbf{g} \cdot \mathbf{g}$
- Correction des biais :  $\hat{m} = \frac{\mathbf{m}}{(1-\omega_1)}$   $\hat{v} = \frac{\mathbf{v}}{(1-\omega_2)}$
- Calcul de mise à jour :  $\Delta\theta = -\epsilon * \frac{\hat{m}}{\sqrt{\hat{v}}+\sigma}$
- $\theta = \theta + \Delta\theta$
- fin tant que

Ensuite, par intuition, on a décidé de tester une méthode d'optimisation de second ordre en pensant que peut-être on obtiendrait de meilleures reconstructions de l'image avec moins d'itérations qu'avec la méthode précédente et le même temps de calcul.

L'algorithme **LBFGS** ou BFGS à mémoire limitée est basé sur la méthode de Newton :

$E_{\theta} = E_{\theta_0} + (\theta - \theta_0)^T \nabla_{\theta} E_{\theta_0} + \frac{1}{2}(\theta - \theta_0)^T \mathbf{H}(\theta - \theta_0)$ , cette méthode donne la règle de mise à jour suivante :  $\theta^* = \theta_0 - \mathbf{H}^{-1} \nabla_{\theta} E_{\theta_0}$ . La difficulté de la méthode se trouve dans le calcul de l'inverse de l'hesienne  $\mathbf{H}$ .

En faisant des expériences, on a aperçu que le temps de calcul de cette dernière méthode dépassait considérablement celui de la première méthode.

Dans le code qui nous a été fourni, l'implementation de la méthode LBFGS est précédée par 100 itérations réalisés avec l'algorithme ADAM. Pour réaliser des comparaisons qualitatives et quantitatives, on a travaillé sur la base de 800 itérations.

Dès figures 22 et 23 on observe que avec le même nombre d'itérations, la première méthode génère une image qui ressemble plus à l'image objectif cherché que l'image générée par la deuxième méthode.

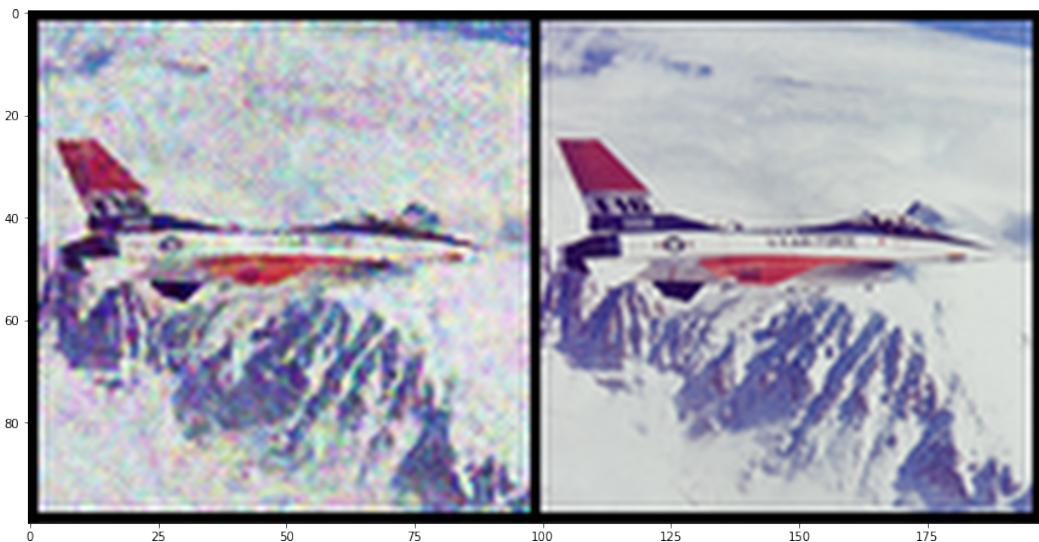


FIGURE 22 – Débruitage avec ADAM sur 800 itérations :image générée(PSRN :24.08) et image objectif

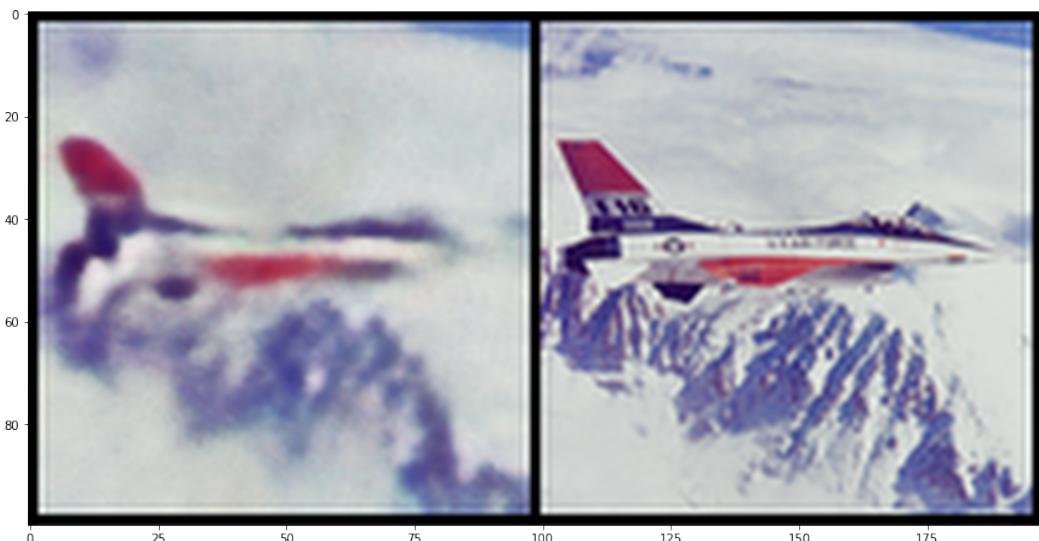


FIGURE 23 – Débruitage avec LBFGS-ADAM sur 800 itérations :image générée(PSRN :22.44) et image objectif

En regardant les figures 24 ,25 et 26 on peut voir que les courbes issues de l'optimisation LBFGS ont toutes un plateau après l'itération 100 ce qui peut signifier qu'on a retrouvé un minimum local ou tout simplement que l'algorithme LBFGS n'est pas adapté au réseau.

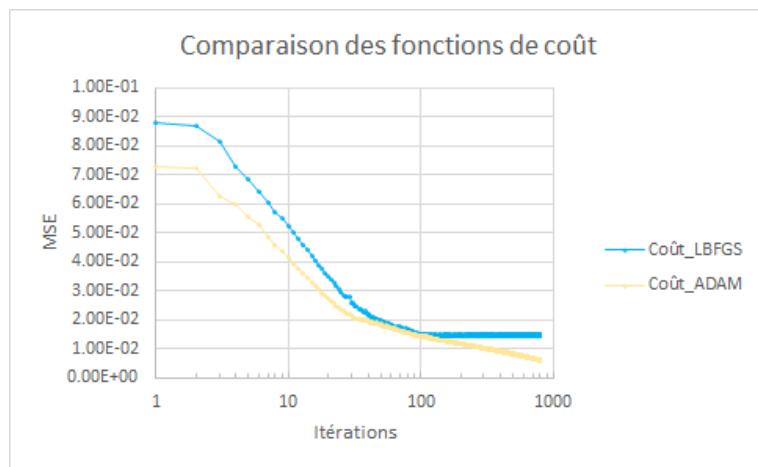


FIGURE 24

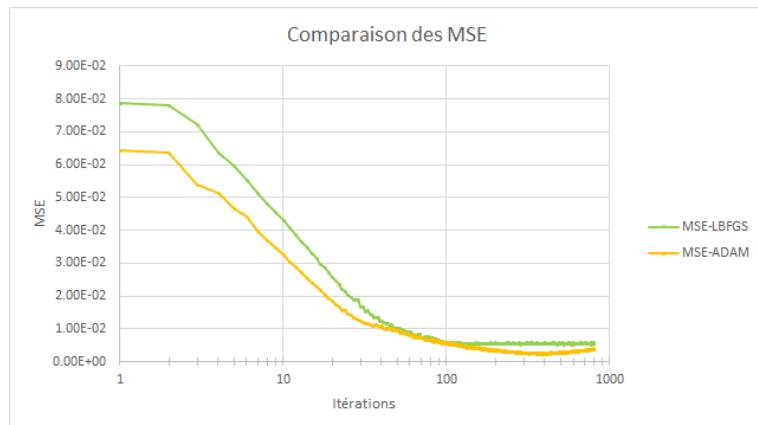


FIGURE 25

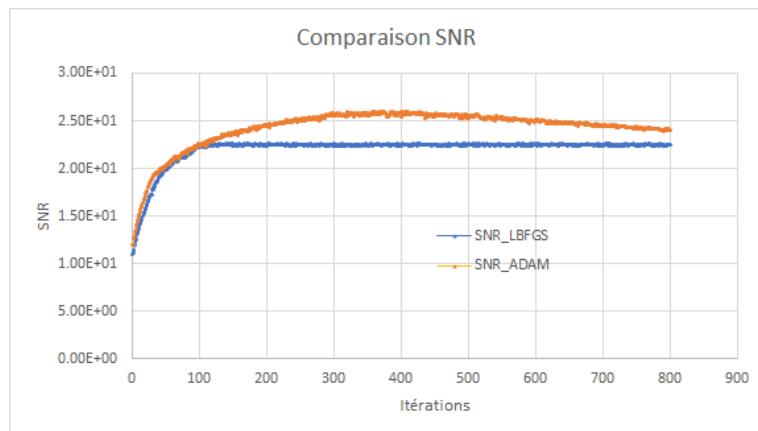


FIGURE 26

## 3.5 Robustesse de la méthode

### 3.5.1 Performances sur des images contenant du bruit réel

Afin d'évaluer la robustesse du débruitage par cette approche aveugle, on s'est proposé d'effectuer une série de tests sur des images présentant du bruit naturel. En effet, toutes les expériences faites précédemment partent d'une image propre dite la "ground truth" à laquelle on rajoute du bruit gaussien additif (AWGN). Même si ce type de bruit modélise bien le bruit qu'une image numérique pourrait contenir, il est tout de même peu exhaustif.

Comme le montre la figure 27, au bout de 1000 itérations nous avons obtenu des images parfaitement débruitées.

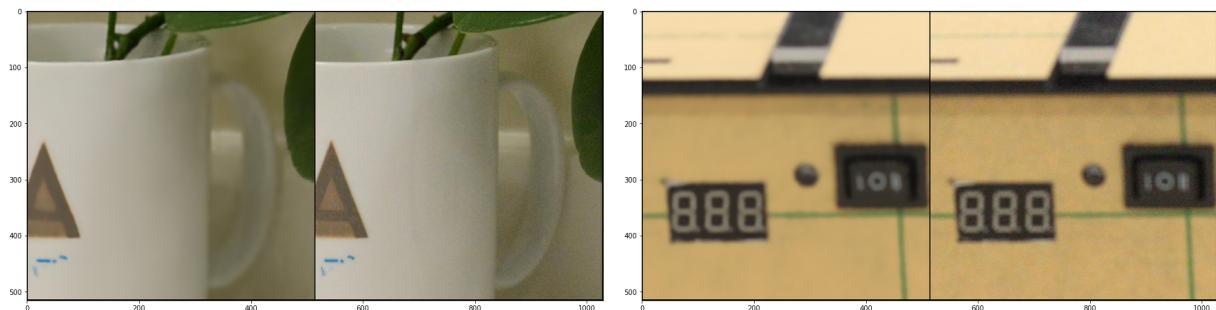


FIGURE 27 – Débruitage d'images contenant du bruit non synthétique (A gauche : l'image débruitée)

## 4 Exécution sur Google Colab

Comme on n'avait pas des ordinateurs trop performants, ni des GPUs disponibles, le code prenait beaucoup de temps à exécuter. Une solution possible pour cela c'est de lancer le code sur la plateforme Google Colaboratory ([colab.research.google.com](https://colab.research.google.com)), un environnement de notebook Jupyter en ligne. Pour ce faire, il suffit d'y télécharger le fichier principal du projet (`denoising.ipynb`) avec des petites modifications.

Il faut simplement y ajouter quatre lignes de code, à savoir :

```
% au tout debut du code :
! git clone https://github.com/DmitryUlyanov/deep-image-prior.git

% dans les imports :
import sys
sys.path.append('/content/deep-image-prior')
```

```
% apres les imports :  
os.chdir( 'deep-image-prior' )
```

Avec la plateforme, on remarque une exécution beaucoup plus vite du code, dûe au fait de qu'on peut s'utiliser de leurs GPUs, notamment CUDA. Malheureusement, on ne l'a découverte qu'à la toute fin du projet, donc on ne l'a pas pu bien explorer.

## 5 Conclusion

Ce petit manuel résume notre travail exploratoire de la nouvelle méthode de débruitage proposée par l'article référence Deep Image Prior.

Si la méthode est intéressante, c'est parce que les performances de débruitage se sont montrées robustes face à des images présentant différents niveaux et différent types de bruits. Cependant, elle présente le défaut d'être coûteuse en terme de calcul et donc elle n'est pas adaptée à une tâche de restauration de plusieurs images en série comme le traitement vidéo.

Certaines déformations appliquées à l'entrée du réseau de neurones ont soulevé l'idée qu'une optimisation de l'algorithme est envisageable : Si l'amélioration de la qualité de débruitage par optimisation par gradient descendant sur l'entrée a débouché sur un échec les manipulations de certains hyperparamètres faites dans une optique de réduction de la complexité de la méthode ont donné des résultats prometteurs.

Toutefois, l'effort de réduction doit être opéré savamment. Car, même si réduire les dimensions de Z et la profondeur du réseau (en terme de couches convolutives) ont diminué drastiquement la charge calculatoire tout en préservant la qualité du débruitage, la suppression des couches de saut (skip connections) s'est laissée faire au prix d'une image débruité mais peu naturelle (présentant des artéfacts) .

Pour le peu d'expériences réalisées, faute d'environnement adapté à l'exécution de ce genres d'algorithmes (GPUs,TPUs), nous étions capable de vérifier que l'algorithme d'optimisation ADAM demeure celui qui rassemble les meilleures performances en termes de vitesse de convergence et de coût de calcul. Mais, nous regrettons ne pas pouvoir affirmer que le réseau réduit n'est pas "overfitté" sur l'image test utilisée pour les expériences.

Enfin le projet nous a été une opportunité durant laquelle nous avons appréhendé les notions de réseaux de neurones. Nous nous sommes également familiarisés avec Pytorch, une framework dédiée à l'implémentation de ses algorithmes qui présente le défaut de la complexité de la visualisation des poids appris comparé à Tensorboard sur Tensorflow chose qui a limité notre interprétation de certains phénomènes observés.

Si nous sommes satisfaits des résultats obtenus nous sommes au regret de ne pas avoir pu approfondir davantage notre étude en raison des limitations matérielles sur nos ordinateurs personnels, si une piste utilisant le service gratuit Colaboratory de Google est proposé nous espérons bien que l'école se chargerait de prévoir un environnement propice d'exécution de ce type d'algorithmes.