Denoising with Deep Learning

Thibault Pirotte

December 2021

1 Introduction

Le denoising est un des principaux champs d'études du traitement des images. Il en existe de nombreuses méthodes avec chacune leur avantages et leur inconvénient: méthode variationnelles, filtrage local (moyenne) et non local (médiane), transformation de Fourier... Nous allons dans ce TP nous intéresser aux débruitages avec Deep Learning en comparant notamment deux réseaux de neurones.

2 Question 1

Dans ce bout de code, on définit la classe *NoisyBSDSDataset* qui correspond en fait au Dataset sur lequel on va entrainer les réseaux. Cette classe à plusieurs attributs:

- Le mode: train ou test selon l'utilisation qu'on veut faire du dataset
- La taille des images
- Le sigma, pour régler la puissance du bruit que l'on rajoute
- Imagedir/files qui indique le chemin vers le repertoire contenant les images

On associe a cette classe plusieurs méthodes:

- len: retourne le nombre d'image du dataset
- repr: retourne une chaine de charactères indiquand les différents attributs (mode, taille des images, sigma...)
- getitem: renvoie l'image à l'index idx ainsi que son équivalent bruitée avec un bruit gaussien aléatoire plus ou moins fort en fonction de sigma.

Figure 1: Image du dataset, avec sa version "clean" et sa version bruitée (sigma = 60)

3 Question 5

On peut retrouver dans le repertoire *Denoising1* deux fichiers. Le premier est un fichier texte appelé *config.txt*, ce fichier contient toutes les informations liées au réseaux. On peut voir par exemple son architecture, l'optimiseur utilisé, le learning rate... Le deuxième fichier "checkpoint.pth.tar" est lui beaucoup plus lourd. Il s'agit en fait d'une sauvegarde de tous les paramètres du réseaux de neurones après un certain nombre d'itérations. Par exemple ici, étant donné que l'on a pas encore entrainé le réseau, ce fichier contient les paramètres initiales. Si on entraine le réseau, alors ce fichier sera modifié (au contraire du fichier txt qui est lui invariant). Il permet donc de stocker l'état du réseaux, et si on entraine le réseau, on repartira de cet état.

4 Question 7 - Résultats du réseau DnCNN

Maintenant regardons les résultats après avoir entrainé le réseau 30 fois.

Figure 2: Débruitage avec DnCNN après 30 itérations)

On a bien réussi à enlever une bonne partie du bruit mais il subsiste néanmoins quelques artefacts et les textures semblent peut-être plus grossières que dans l'image clean. On pourra remarquer que l'on a pas d'effets de flou comme l'on peut en rencontrer quand on utilise d'autres méthodes de débruitage.

Maintenant regardons les courbes de la fonction de perte ainsi que de la PSNR:

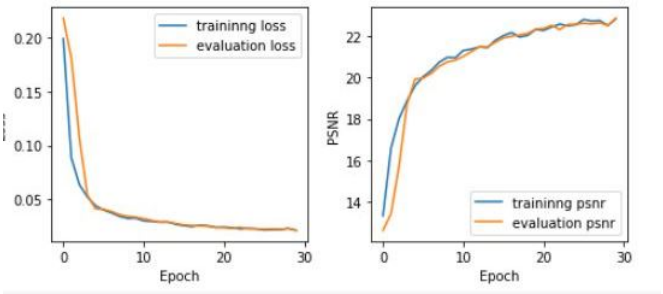


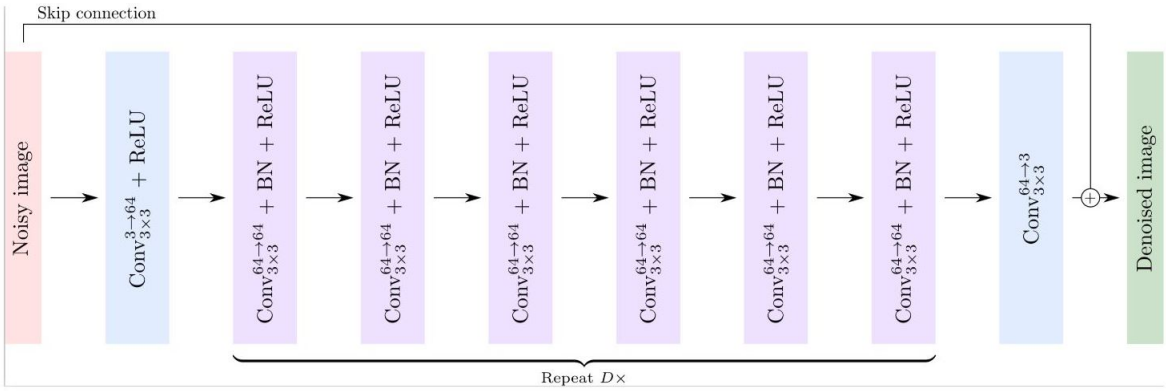
Figure 3: Gauche: Evolution de la MSE Droite: Evolution de la PSNR)

On pourra noter que on a effectivement une diminution de l'erreur quadratique, et une augmentation de la PSNR, ce qui prouve bien que le réseau fait ce qui est attendu de lui. Néanmoins, après 30 itérations la progression semble stagner, ce qui peut faire penser que faire plus d'itérations n'est pas utile. Enfin, dernier point intéressant, les courbes pour les sets d'entrainements et de validations ont quasiment les même valeurs, ce qui peut faire penser que l'on a pas d'overfitting.

Figure 4: Résultats sur une autre image

5 Question 8

On cherche à connaitre le nombre de paramètres modifiables pour le réseau. Rappelons l'architecture de celui-ci:



Prenons tout d'abord le nombre de paramètres liés au couches de convolution. En comptant le nombre de bias ce nombre sera

$$\begin{aligned} n &= (3*3*3+1)*64 + D*(3*3*64+1)*64 + (3*3*64+1)*3 \\ &= 1792 + D*36928 + 1731 \end{aligned} \tag{1}$$

Le nombre de paramètres pour chaque couche de Batch Normalization est egale à 2 fois la profondeur de la couche de convolution précédente. Dans notre cas, on aura donc

$$\begin{aligned} m &= D*(64*2) \\ &= D*128 \end{aligned} \tag{2}$$

Par conséquent, le nombre total de paramètres en prenant D = 6 est:

$$\begin{aligned} N &= m + d \\ &= 1792 + 6*36928 + 1731 + 6*128 \\ &= 225091 + 768 \\ &= 225859 \end{aligned} \tag{3}$$

Enfin, nous cherchons à connaître la taille du receptive field, c'est à dire le nombre de pixels dans l'image d'entrée qui influence un pixel de l'image de sortie.
Prenons un cas où on a un réseau avec juste une couche de convolution $3*3$. Alors, un pixel du résultat sera influé par $3*3$ pixels. Maintenant, si on rajoute une autre couche de convolution $3*3$, ce même pixel sera influencé par tous les pixels qui influencent les $3*3$ pixels précédemment. Le réceptive field sera donc $5*5$.
En fait, si on considère le réceptive field comme un carré de pixels de taille $n*n$, alors, à chaque couche de convolution de taille $3*3$, on augmente n de 2.
Par conséquent, pour notre réseaux de neurones le réceptive field sera de taille $n*n$, avec

$$n = 1 + 2 + 2 + 2*D \tag{4}$$

Par exemple, si on prend $D = 6$, alors $n = 17$, et le receptive field est un carré de taille $n * n = 17 * 17 = 289$.

Selon certains papiers, un réseau débruiteur est efficace si le receptive field est égale à $33 * 33$. Cela impliquerait ici que D soit égale 14, or cela risque d'augmenter de manière considérable le temps d'entrainement du réseau.

6 Question 10

On cherche maintenant à tester un autre réseaux, inspirés par l'architecture U-Net. Regardons les résultats:

Figure 5: Débruitage avec UDnCNN après 30 itérations)

Le résultat du débruitage en utilisant les 2 réseaux semblent très proche. Cette impression semble se confirmer quand on regarde les courbes de la MSE et PSNR pour l'UDnCNN:

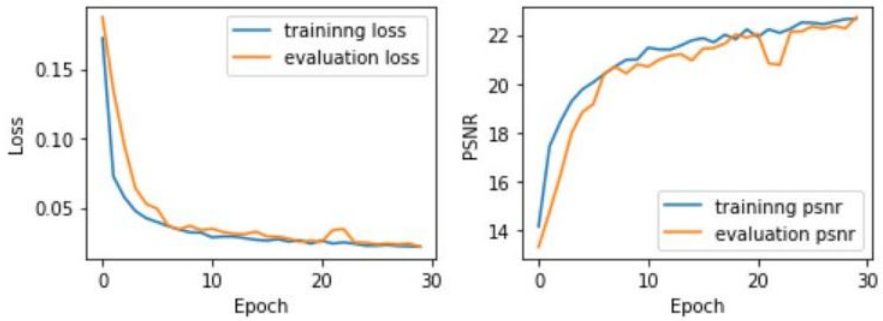
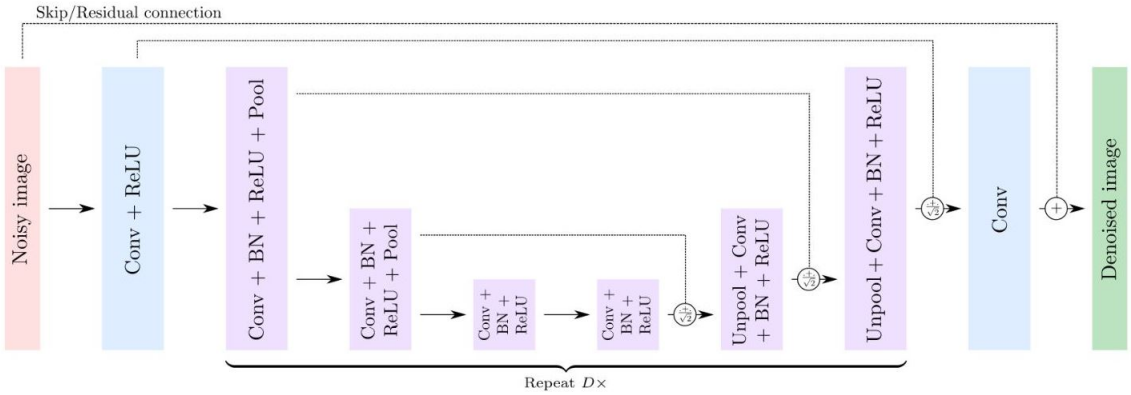


Figure 6: Gauche: Evolution de la MSE Droite: Evolution de la PSNR)

Les valeurs obtenues sont très proches de celles qu'on avait pour le premier réseau, ce qui permet de confirmer que les deux ont un niveau de performance très proche. Petit détail qui peut cependant être important, le réseau UDnCnn (le deuxième) semble un petit peu plus rapide.

7 Question 11

On cherche à estimer le nombres de paramètres de UDnCNN. Rappelons son architecture:



La principale différence par rapport au réseau précédent est l'utilisation de Max Pooling et de Unpool pour réduire puis reformer l'image. Or, ces couches-ci ne possèdent pas de paramètres variables, et les couches convolutives et de Batch Normalisation sont identiques que pour DncCNN. Par conséquent, le nombre de paramètres de UDnCNN est le même que pour DnCNN: 225859.

Maintenant, nous souhaitons calculer le receptive field de UDnCNN. Nous savons comment évolue le receptive field en fonction des couches de convolutions, mais qu'en est-il pour les couches de Max Pooling?

Quand on applique le Max Pooling à une image, alors un pixel de sortie dépendra de $2 * 2 = 4$ pixel de l'image d'entrée. Il faut également noter que chaque pixel de l'image d'entrée influe sur un seul et unique pixel de l'image de sortie (à la différence d'une couche de convolution). Par conséquent, $n * n$ pixels de l'image de sortie seront définis par $2n * 2n$ pixels.

Maintenant, on peut facilement déduire la taille du receptive field de la partie contractante de UDnCNN en regardant son architecture. Si on prend $D = 6$, alors celui-ci sera un carré de taille $n*n$ avec $n = (1+2+2)*2+2)*2+2)+2 = 28(5)$
La deuxième partie du réseau consiste en une série de concaténations, de couches UnPool, et de couches de convolution $3 * 3$. Les concaténations et les couches UnPool ne modifient pas le receptive field. Il faut donc juste rajouter à n 2 fois le nombre de couches de convolutions qui restent.
Par conséquent $n = 27 + 2 * 3 = 34$

Donc le receptive field est de taille $34 * 34 = 1156$

On notera que UDnCNN a un receptive field beaucoup plus étendu que DnCNN pour un même nombre de paramètres, ce qui améliore théoriquement la qualité du débruitage sans augmenter le temps de calcul. Les dimensions dépassent d'ailleurs $33 * 33$ ce qui est censé nous garantir que le réseau est efficace. Ce gain est toutefois relativisé par le fait que l'on perd de l'information dans la partie contractante.

8 Question 12

Nous avons a chaque fois choisi comme fonction de perte l'erreur moyenne quadratique (c'est à dire la norme L2) dans les exemples précédents. Maintenant, la question qui peut se poser est que se passe-t'il lorsque l'on utilise une autre fonction de perte?

Comment choisir cette fonction? Pour un effet similaire, on utilisera plutôt une fonction avec une forme similaire à la norme L2. Un choix évident ici est d'utiliser la norme L1, aussi appelé SAD (sum of absolute differences).

Testons le réseaux avec la SAD comme fonction de perte:

Figure 7: Loss = SAD ou Norme L1)

Les résultats semblent plutôt satisfaisant et très proches de ceux qu'on avait précédemment. Le débruitage semble même un tout petit peu plus poussé, mais avec néanmoins des artefacts un peu plus présents. Pour comparer les résultats, on peut éventuellement regarder la PSNR (attention néanmoins car cet indicateur dépend de l'erreur quadratique et non pas de l'erreur absolue).

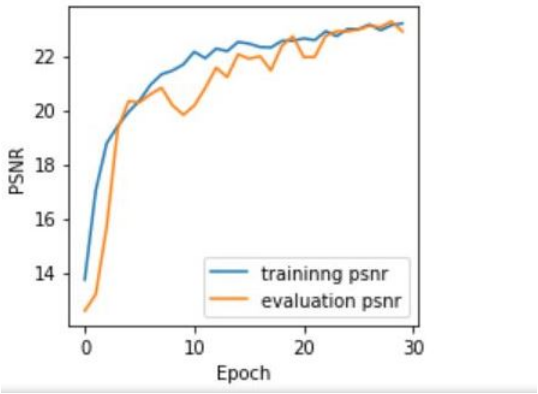


Figure 8: Evolution de la PSNR avec la SAD

On retrouve comme prévu des valeurs proches de ce que l'on avait précedemment (on a utilisé ici l'architecture DnCNN).

Attention aux choix de la fonction de perte. En effet, certaines fonctions ne sont pas adaptées pour certains problèmes. Par exemple la Cross-Entropy n'est pas du tout adaptée au débruitage.

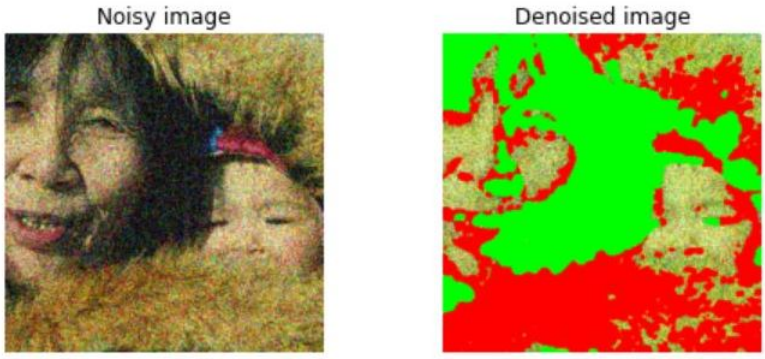


Figure 9: Tentative de débruitage en utilisant la cross entropy

La cross-entropy est en effet une fonction de perte principalement utilisé pour les problèmes de classification.

9 Questions additionnelles

9.1 Qu'est-ce que l'opérateur gradient

L'opérateur gradient est une généralisation de la notion de dérivées. Il s'agit d'un vecteur contenant l'essentiel des dérivées premières de la fonction. Par

exemple, dans le cas d'une application F allant de $R^n à R$ alors son opérateur gradient s'écrira: $\nabla F = (\frac{\partial F}{\partial x1}, ..., \frac{\partial F}{\partial xn})$.

9.2 Différence entre la norme L2, la distance euclidienne et la SSD

Dans le cas discret, la norme L2 associé à une matrice ou a un vecteur correspond à la racine carré de la somme de l'ensemble des carrés des élements de la matrice.

La distance euclidienne est une mesure de distance entre deux matrices de même taille. Il s'agit en fait de la norme L2 de la matrice résultant de la différence entre les deux matrices dont on cherche à calculer la distance.

$$DistanceEuclidienne(A, B) = ||A - B||_2 \tag{6}$$

La Sum of Squared Differences est juste le carré de la distance euclidienne, donc le carré de la norme L2. Par conséquent, il s'agit juste d'une somme de carrés, d'où le nom de Sum of Squared Differences.

9.3 Différence entre Corrélation et Convolution

Dans le cas du traitement des images, la convolution consiste à appliquer sur un pixel la somme des pixels voisins auquels on a appliqué à chacun un poids. On parle alors de filtrage linéaire (exemple: filtre moyennant).

La corrélation elle, est une mesure de la similarité entre 2 signaux/images.