

[HAI927I] Projet Image - Compte Rendu n°5

Projet #12.3 : Débruitage d'images

1. Mise en place d'analyse / visualisation des résultats

Nous avons continué les travaux de la semaine dernière. Nous avons créé un nouveau programme qui génère une courbe grâce à gnuplot selon une mesure de comparaison (PSNR, SNR, SSIM, RMSE) et une variable choisie par l'utilisateur.

Dans notre cas, il y a toujours plusieurs variables en jeu (par exemple : des images sont bruitées avec un filtre gaussien utilisant une moyenne M et un écart-type E puis débruitées avec un filtre médian ayant une fenêtre de taille T). Il faut donc fixer une valeur pour toutes les autres variables afin de tracer la courbe.

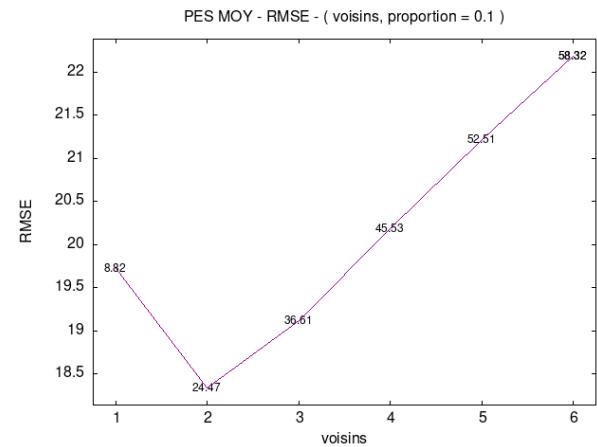
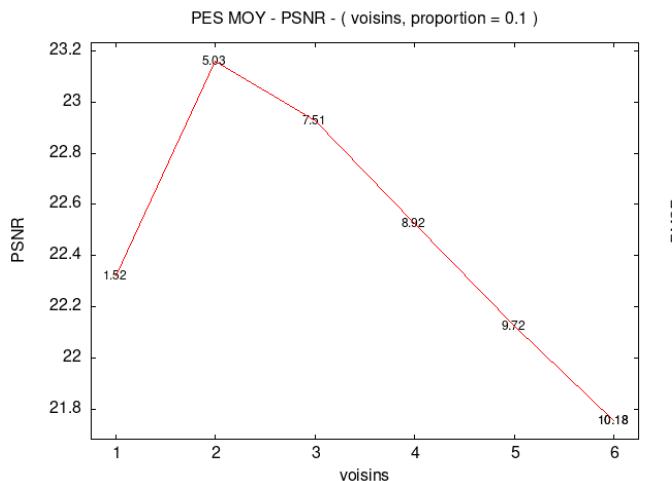
Voici le fonctionnement des scripts permettant de générer les données évaluant les résultats obtenus ainsi que la variation de la qualité des résultats selon la valeur des paramètres (et ça pour des images en nuances de gris et en couleurs) :

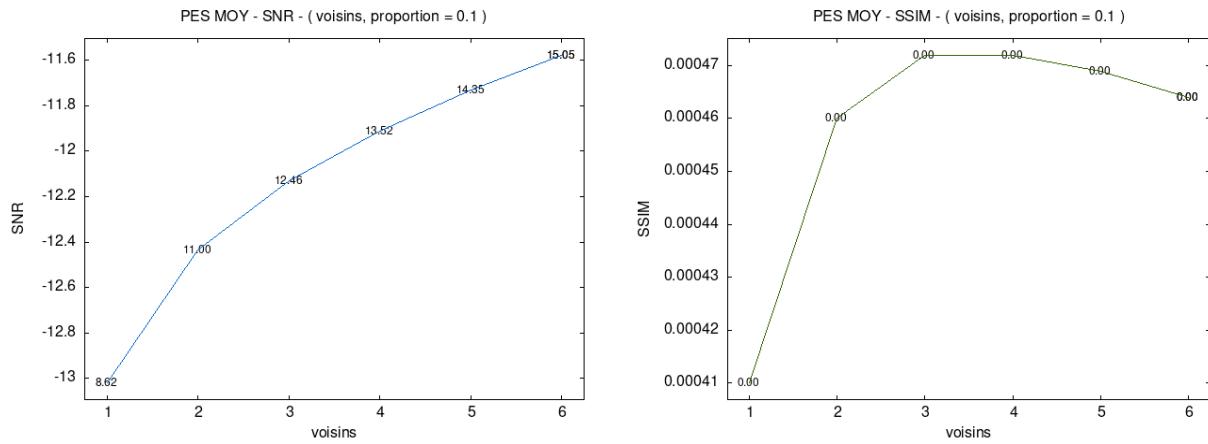
- On renseigne une base de données dans laquelle toutes les images utilisées pour mesurer l'efficacité des méthodes de débruitage se trouvent selon les méthodes de bruitage. Pour ce faire, on choisit la base de données CID22, composée de 250 images de taille 512 x 512. Pour être exploitable par nos programmes, on convertit toutes les images en ppm et en pgm en utilisant une de nos bases de code appelée "Mosaïque". On divise ensuite ces images en deux bases de données : une base de données pgm et une base de données ppm. Voici 3 images de la base de données CID22 :



- Pour chaque bruit encodé (poivre et sel, gaussien, impulsif, speckle et poisson), nous générerons un ensemble d'images. Un ensemble d'images contient toutes les images de CID22 modifiées. Pour chaque bruit, ces ensembles d'images sont bruitées avec différents paramètres.
- Pour méthodes de débruitage encodé (moyenieur, médian, gaussien, gradient, moyenieur pondéré et non local means), on débruite chaque ensemble d'images précédemment généré avec différents paramètres.
- Pour chaque couple (bruit, méthode de débruitage), on calcule le PSNR, SNR, SSIM et RMSE entre l'image de base et l'image bruitée débruitée, pour toutes les images de l'ensemble. A partir de ces valeurs, on calcule la moyenne et la variance de ces résultats. La moyenne et la variance sont stockées dans un fichier texte dédié au tuple (bruit, méthode de débruitage, {PSNR|SNR|SSIM|RMSE}) avec les différents paramètres utilisés.
- Pour chaque tuple (bruit, paramètres du bruit, méthode de débruitage, paramètres de la méthode de débruitage, {PSNR|SNR|SSIM|RMSE}), on crée plusieurs courbes comparant la moyenne du {PSNR|SNR|SSIM|RMSE} et un des paramètres. On doit fixer la valeur de toutes les autres. La variance du {PSNR|SNR|SSIM|RMSE} est affiché sur la courbe.

Voici des exemples de courbes. Ici, on cherche à voir le changement du {PSNR|SNR|SSIM|RMSE} quand on débrute une image avec un filtre moyenieur ayant différentes tailles de fenêtre (voisins) sachant que cette image a été bruitée par un bruit poivre et sel d'une intensité 0.1 :





2. Mise en place du réseau de neurones

Après avoir discuté avec M. Dibot qui nous a conseillé d'utiliser un réseau de neurones déjà entraîné, nous avons décidé de mettre de côté le CBDNet car nous n'en avons pas trouvé qui était déjà entraîné.

Nous avons essayé de partir sur un réseau de type RIDNet mais les résultats obtenus n'étaient pas très satisfaisants, et en plus il ne s'appliquait que sur les images en couleur.

Finalement, nous avons trouvé un réseau de type Transformer qui s'appelle le Restormer et qui nous donne des résultats très satisfaisants sur les images en noir et blanc et en couleur. Voici le lien du dépôt github de ce réseau :

<https://github.com/swz30/Restormer/tree/main/Denoising>.

Le Restormer que nous avons trouvé et adapté permet de restaurer des images hautes résolutions.

Il utilise un modèle déjà entraîné dans un fichier “real_denoising.pth” et la classe du modèle dans “restormer_arch.py”.

Voici l'explication du code “preEntrainement_Restormer.py” que nous avons créé pour débruiter nos images :

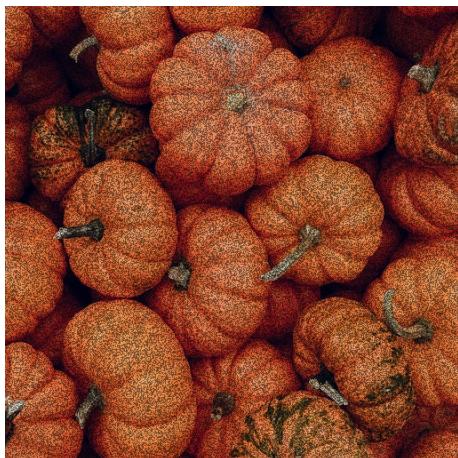
- Il va récupérer les poids et un certain nombres de paramètres stockés dans le modèle “real_denoising.pth”
- Il va charger l'architecture du modèle : il va exécuter le fichier “restormer_arch.py” et charger la classe Restormer du modèle avec les paramètres définis précédemment
- On va ensuite entrer dans la boucle principale :
 - Les fichiers d'images dans le répertoire d'entrée sont traités un par un
 - Chaque image est chargée, prétraitée (notamment redimensionnée) et ensuite soumise au modèle Restormer
 - La sortie du modèle est ensuite enregistrée dans le répertoire de sortie après quelques opérations de post-traitement (limiter les valeurs de pixels, faire des conversions, etc...)

Voici nos images de base avant bruitage et débruitage :

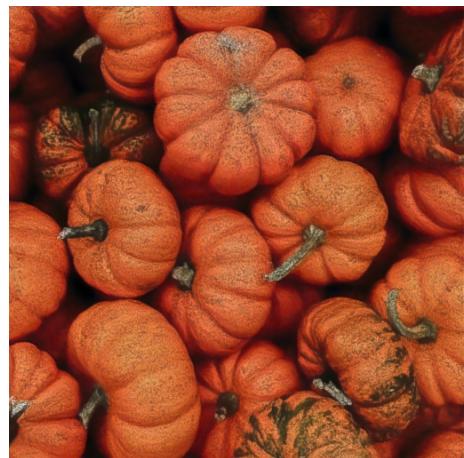


Voici les résultats obtenus :

Bruit	Débruitage avec le Restormer
Bruit Gaussien (moyenne = 0, écart-type = 25)	 
PSNR entre l'image bruitée et l'image de base = 20.36	PSNR entre l'image débruitée et l'image de base = 26.66
Bruit Impulsif (facteur = 50)	



PSNR entre l'image bruitée et l'image de base = 15.28



PSNR entre l'image débruitée et l'image de base = 21.31

Bruit Speckle (intensité = 25)

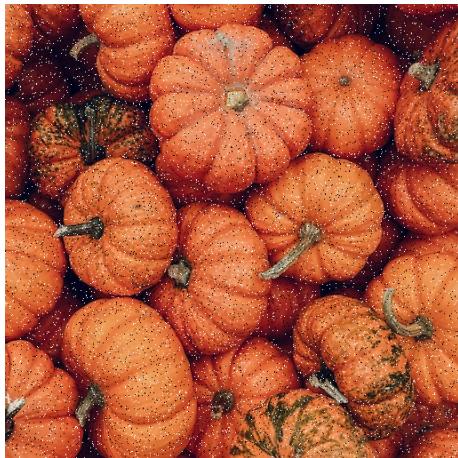


PSNR entre l'image bruitée et l'image de base = 24.83



PSNR entre l'image débruitée et l'image de base = 29.76

Bruit Poivre et Sel (proportion = 0.05)



PSNR entre l'image bruitée et l'image de base = 17.65	PSNR entre l'image débruitée et l'image de base = 25.82
---	---

3. Fonctionnement de l'entraînement du Restormer

Nous avons étudié le code réalisé pour l'entraînement du Restormer.

- Pour commencer, ils téléchargent la base de données de l'entraînement qui est une base SSID (Smartphone Image Denoising Dataset)
- Ensuite, ils appellent un algorithme qui va générer des patchs d'images à partir des images d'entraînement
- Ils lancent ensuite le script d'entraînement “train.sh” avec en paramètre le fichier “RealDenoising_Restormer.yml”
 - train.sh : lance l'entraînement sur plusieurs gpu avec pytorch avec le fichier de configuration “RealDenoising_Restormer.yml”
 - RealDenoising_Restormer.yml : contient beaucoup de paramètres pour définir le modèle, le nombre d'itérations, les paramètres de validation, les paramètres d'optimisations, etc...

Le Restormer fonctionne en encodant les caractéristiques de l'image bruitée, en utilisant un bloc de restauration pour modéliser la relation entre les caractéristiques et les informations de bruit, puis en décodant pour produire une image débruitée.

Il utilise l'attention multi-têtes, concept couramment utilisé dans les réseaux de type Transformer, qui permet d'utiliser plusieurs têtes d'attention indépendantes simultanément et chaque tête va générer une représentation différente de l'entrée.
Les sorties des différentes têtes d'attention sont concaténées et passent par une opération linéaire (nouvelle projection) pour obtenir la sortie finale de l'attention multi-têtes.

La multi-head attention a plusieurs avantages :

- Capturer des Relations à Différentes Échelles : Chaque tête peut se spécialiser dans la capture de relations à différentes échelles spatiales ou temporelles
- Améliorer la Robustesse : En utilisant plusieurs têtes, le modèle peut apprendre différentes représentations de la même entrée, ce qui améliore sa robustesse et sa capacité à généraliser
- Réduire le Surapprentissage : L'utilisation de plusieurs tête peut aider à réduire le surapprentissage, car chaque tête apprend une représentation indépendante

4. Objectifs de la semaine

- Produire les données d'analyse
- Commencer l'application QT
- Faire le poster scientifique