

[HAI927I] Projet Image - Compte Rendu n°3

Projet #12.3 : Débruitage d'images

1. Algorithmes de bruitage

Nous avons amélioré les algorithmes de bruits que nous avions faits la semaine dernière et réalisé ceux qui manquaient. Nous avons aussi réalisé ces bruits sur des images en couleur.

- Bruit Gaussien

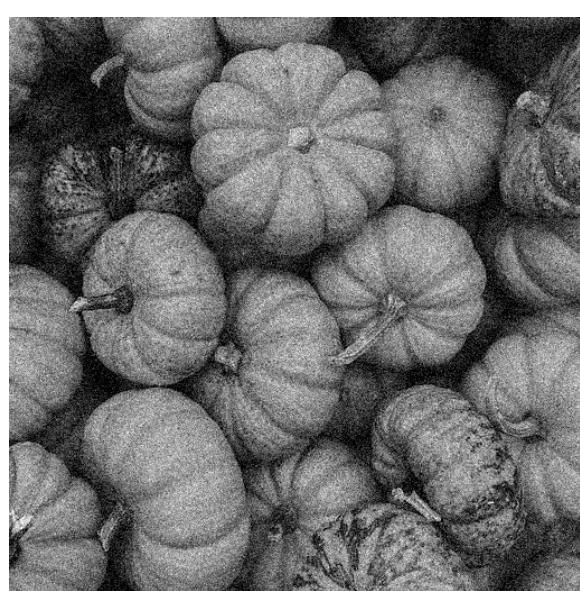
Pour le bruit Gaussien, la semaine dernière nous avions des résultats étranges avec beaucoup de pixels blancs. Nous avons corrigé ce problème qui n'était dû qu'à une toute petite erreur dans le code.

L'algorithme fonctionne de la façon suivante :

Chaque pixel de notre image va être additionné avec un nombre aléatoire qui suit une distribution gaussienne.

moyenne = 0, écart-type = 10

moyenne = 0, écart-type = 25



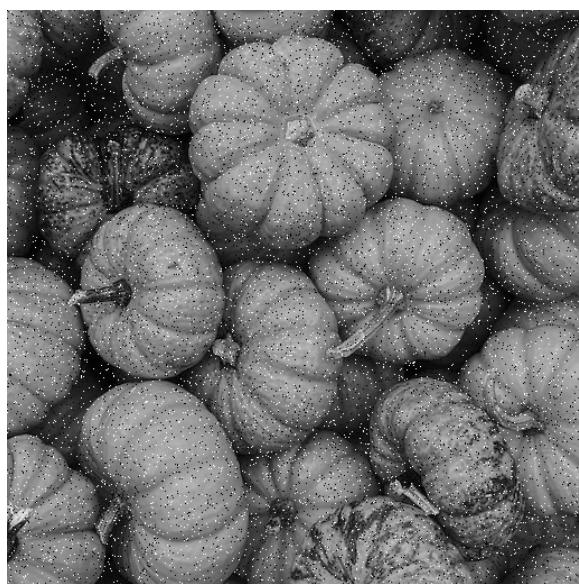
- Bruit Poivre et Sel

Au niveau du bruit Poivre et Sel, nous avions déjà un algorithme qui fonctionnait bien la semaine dernière, nous l'avons donc modifié pour qu'il prenne en compte les images en couleur.

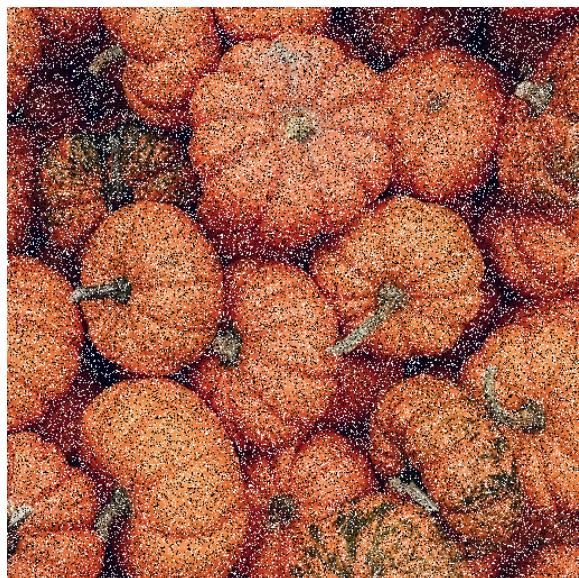
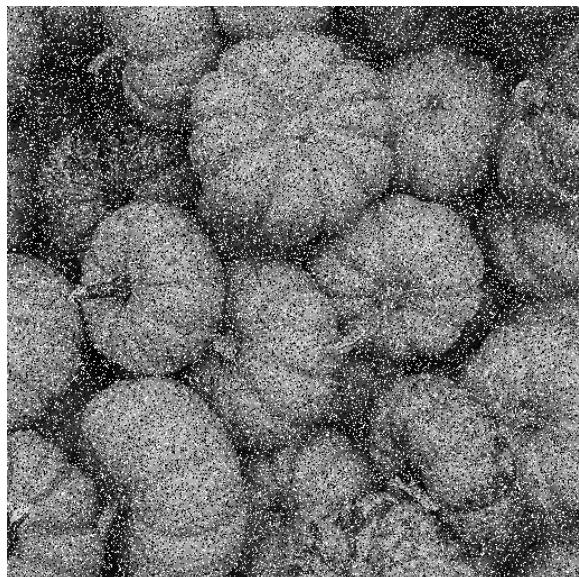
L'algorithme fonctionne de la façon suivante :

Pour chaque pixel de notre image, on va générer un nombre aléatoire entre 0 et 1. Si ce nombre est inférieur à la proportion divisée par 2 alors on met le pixel à 0 et si le nombre est supérieur à $1 - \text{la proportion divisée par } 2$, alors on met le pixel à 255 et sinon le pixel garde sa valeur.

proportion = 0.05	proportion = 0.10
-------------------	-------------------



proportion = 0.25



- Bruit Impulsif

L'algorithme fonctionne de la façon suivante :

On va prendre en entrée un facteur puis on va, pour chaque pixel, générer un nombre aléatoire entre 0 et 100. Si le nombre aléatoire est inférieur au facteur donné alors on divise la valeur du pixel par 2, sinon le pixel garde sa valeur.

Nous avons fait notre algorithme comme cela pour que plus le facteur est grand, plus on a de bruit sur l'image.

facteur = 10	facteur = 25
--------------	--------------



facteur = 50

facteur = 75

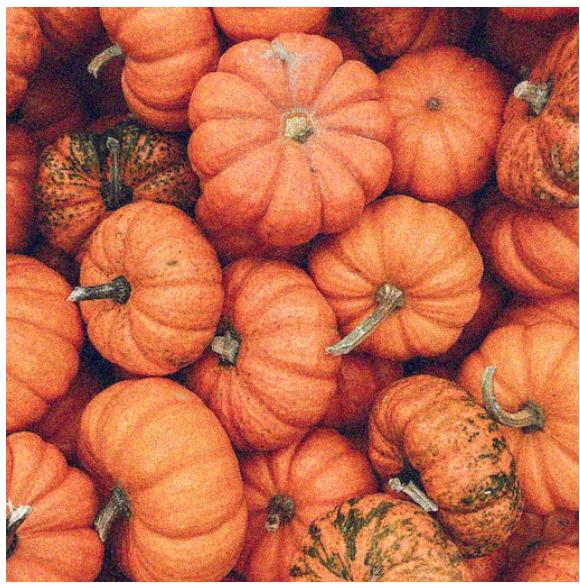
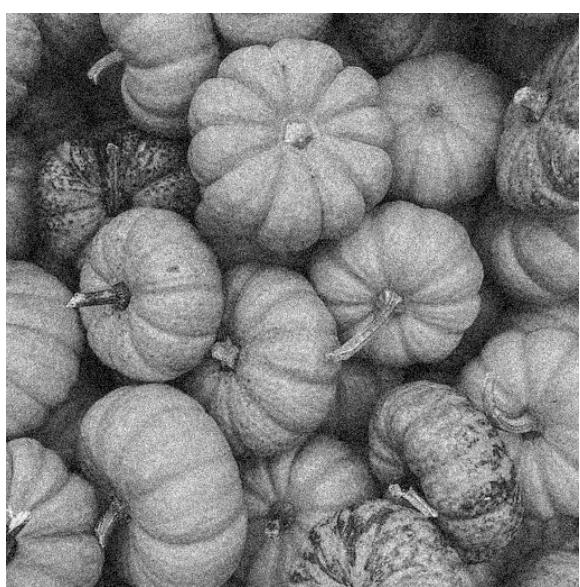


- Bruit de Poisson

L'algorithme fonctionne de la façon suivante :

C'est la même chose que pour le bruit gaussien mais avec la moyenne et l'écart type qui ont la même valeur.

paramètre = 10	paramètre = 20
----------------	----------------



- Bruit de Speckle

L'algorithme fonctionne de la façon suivante :

On a en paramètres une intensité et on va calculer, pour chaque pixel, une valeur aléatoire entre les bornes [-intensité, intensité] et on va l'additionner avec la valeur du pixel actuel.

intensité = 25	intensité = 50
----------------	----------------



intensité = 100

intensité = 200



2. Algorithmes de débruitage

- Non Local Means

On a en entrée plusieurs paramètres :

- sigma, qui correspond à la pondération en fonction de la similarité (un sigma élevé donne plus d'importance aux pixels locaux et réduit l'influence des pixels éloignés)
- h, qui correspond à la taille de la fenêtre de recherche pour trouver des patchs similaires
- tailleFenetre, qui correspond à la taille des patchs

Cet algorithme fonctionne de la façon suivante :

Pour chaque pixel de notre image, nous extrayons un patch correspondant (de taille `tailleFenetre * tailleFenetre`), que nous appelons "patchActuel". Nous initialisons les variables `sommePoids` et `sommePoidsValeurs` à zéro.

Ensute, nous récupérons tous les autres patches de l'image, chacun étant appelé "patchVoisin". Nous calculons la mesure de similarité entre le `patchActuel` et le `patchVoisin` en effectuant la somme des carrés des différences de valeurs entre les deux patchs.

En utilisant la formule $\exp(-(\text{mesureSimilarite} * \text{mesureSimilarite}) / (\sigma * \sigma * h * h))$, nous calculons le poids associé à cette similarité. Nous ajoutons ce nouveau poids à `sommePoids`, et nous ajoutons le produit du poids et de la valeur du pixel voisin (celui associé à `patchVoisin`) à `sommePoidsValeurs`.

Enfin, nous calculons la valeur du pixel correspondant dans l'image de sortie avec la formule : `pixelSortie = sommePoidsValeurs / sommePoids`.

Voici quelques exemple sur les bruits qu'on a fait plus tôt :

Poivre et Sel :

$\sigma = 0.05, h = 10, \text{tailleFenetre} = 2$	$\sigma = 0.05, h = 30, \text{tailleFenetre} = 2$
	
$\sigma = 0.05, h = 30, \text{tailleFenetre} = 4$	$\sigma = 10, h = 10, \text{tailleFenetre} = 2$



Impulsif :

$\sigma = 0.05, h = 10, \text{tailleFenetre} = 2$	$\sigma = 0.05, h = 30, \text{tailleFenetre} = 2$
A grayscale image of a dense cluster of small, rounded objects. The noise level is relatively low compared to the first image.	A grayscale image of the same fruit cluster as above, but with a larger window size (h=30). The noise level is significantly reduced, resulting in a clearer image.
$\sigma = 0.05, h = 30, \text{tailleFenetre} = 4$	$\sigma = 10, h = 10, \text{tailleFenetre} = 2$



Gaussien :

$\text{sigma} = 0.05, h = 10, \text{tailleFenetre} = 2$	$\text{sigma} = 0.05, h = 30, \text{tailleFenetre} = 2$
$\text{sigma} = 0.05, h = 30, \text{tailleFenetre} = 4$	$\text{sigma} = 10, h = 10, \text{tailleFenetre} = 2$



3. Filtres de débruitage

Voici les filtres de débruitage implémentés à ce jour. Chaque bruit prend deux arguments obligatoires : le nombre de voisins que l'on prend en compte pour chaque filtre (distance manhattan) et l'intensité du filtrage. Cette intensité est comprise entre 0 et 1 et on pondère la nouvelle valeur du pixel calculée et l'ancienne valeur du pixel tel que : $\text{ImgOut}[\text{pixel}] = \text{intensite} * \text{NouvelleValeur} + (1 - \text{intensite}) * \text{AncienneValeur}$; L'intensité a une valeur par défaut de 1.

Chaque bruit est implémenté pour les images .pgm et .ppm (sauf le filtre de Wiener utilisant la transformée de Fourier 2D).

Les exemples affichés dans cette partie ne sont pas bruités avant l'application du filtre (c'est juste pour voir l'effet que cela a sur une image).

- Filtre Moyenneur

Ce filtre remplace un pixel par la moyenne de ses voisins et lui-même.

#voisins = 1	#voisins = 4	#voisins = 4, intensité = 0.5
--------------	--------------	-------------------------------



- Filtre Médian

Ce filtre stocke toutes les valeurs du pixel et de ses pixels voisins dans un tableau, trie ce tableau et affecte au pixel la valeur de l'élément au milieu du tableau.

#voisins = 2	#voisins = 3	#voisins = 4, intensité = 0.5

- Filtre Gaussien

Ce filtre utilise une distribution gaussienne en 2D pour créer un filtre faisant une somme pondérée de chaque pixel et de ses voisins. Ce filtre prend deux paramètres en plus : la moyenne et la variance utilisées pendant le calcul du filtre. Si ces dernières ne sont pas renseignées alors la distribution gaussienne est centrée-réduite.

#voisins = 1	#voisins = 3	#voisins = 3, moyenne = -5, variance = 10



- Filtre Wiener comme implémenté dans PTC Mathcad Prime

Ce filtre suit la formule donnée sur le site de documentation de PTC Mathcad Prime. Il prend un argument en plus : la variance du bruit.

#voisins = 4, variance = 100	#voisins = 3, variance = 15, intensité = 0.5	#voisins = 4, variance = 9999999

Ce filtre propose des résultats cohérents seulement quand l'intensité est égale ou inférieure à 0.5 ou quand la variance est très grande. On peut en conclure que ce filtre n'est pas satisfaisant.

- Filtre Wiener avec une transformée de Fourier

Ce filtre a été proposé par Chat GPT lorsqu'on lui a demandé de décrire le filtre de Wiener. Il fait intervenir une transformée de Fourier 2D pour déterminer les coefficients à utiliser dans le filtre. Il prend un argument en plus : la variance du bruit.

#voisins = 1, variance = 1	#voisins = 3, variance = 0.5, intensité = 0.5	#voisins = 2, variance = 2
----------------------------	---	----------------------------



Ce filtre propose des résultats intéressants mais prend beaucoup trop longtemps à s'exécuter malgré les optimisations mises en place, même sur une image de 256 x 256. C'est pour cela qu'il n'a pas été implémenté pour les images en couleurs.

4. Idées de filtres de débruitage

- Pour le bruit poivre et sel : enlever tous les pixels 0 et 255 et faire une moyenne des valeurs (existants) de leur voisins.
- Utiliser la carte de gradient et la seuiller pour déduire les contours de l'image. On applique ensuite un filtre moyen / médian / gaussien sur l'image en prenant seulement les pixels voisins noirs que l'on peut atteindre depuis le pixel courant noir. Vice versa pour les pixels blancs.
- Utiliser l'algorithme k means pour établir des zones ayant la même couleur et appliquer un filtre moyen / médian / gaussien dans ces zones (les pixels d'autres zones ne sont pas pris en compte)
- Utiliser la même logique que la méthode présentée ci-dessus mais avec l'algorithme SNC (super pixels). Cela peut être pertinent car l'espace couleur et les paramètres de chaque algorithmes diffèrent.

5. Objectifs de la semaine

- Implémenter le filtre de Wiener avec la transformée en ondelettes
- S'intéresser au débruitage des images par réseaux de neurones
- Explorer les idées de filtres de débruitage
- Voir pour peut-être paralléliser (CPU ou GPU) la transformée de fourier rapide en 2D