

Christal

Compte rendu n°4

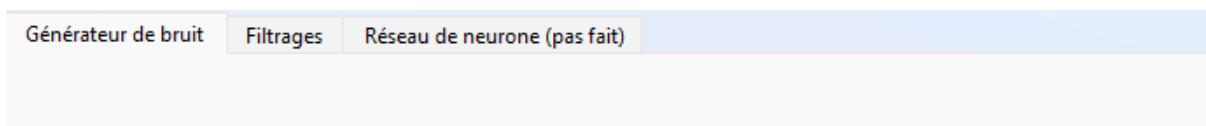
Interface graphique

Cette semaine, nous avons ajusté les éléments de notre interface graphique afin qu'elle soit agréable à configurer pour l'utilisateur.

Dans un premier temps, nous avons mis nos images au bon ratio dans nos QFrames carrées.



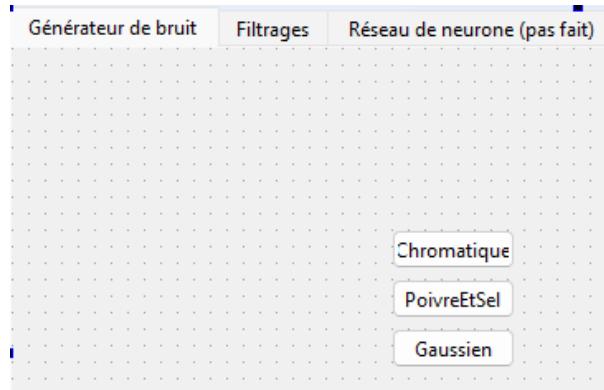
Ensuite, nous avons décidé de rassembler toutes nos méthodes dans un tabWidget afin de les classer par "type".



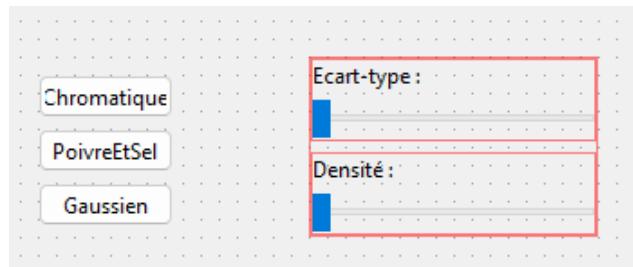
Cela oblige ainsi l'utilisateur à choisir ce qu'il veut faire et ne pas mélanger les différentes fonctionnalités à sa disposition. Nous y rajouterons par la suite la méthode par réseau de neurones.

- Générateur de bruit :

La semaine dernière, nous avions ajouté des boutons qui permettent de générer du bruit. Nous les avons donc déplacés dans le tabWidget.



Après notre conversation avec Nicolas Dibot, durant laquelle nous lui avions montré les fonctionnalités de nos filtres (dont nous parlons dans le point suivant), nous avons suivi sa demande, à savoir, s'il était possible de pouvoir modifier les valeurs des paramètres des générateurs.



Ainsi, nous avons désormais la possibilité de choisir l'écart-type ou la densité du bruit.

- Filtres :

Nous avons ensuite une partie "filtres", qui permet d'avoir le rendu que l'on souhaite directement dans notre application. Nous avons implémenté les différentes variables associées aux fonctions des filtres (radius, diamètre, variation spatiale, etc...). Cela permet ainsi de pouvoir tester directement plusieurs valeurs et avoir une idée du meilleur résultat à l'œil nu.

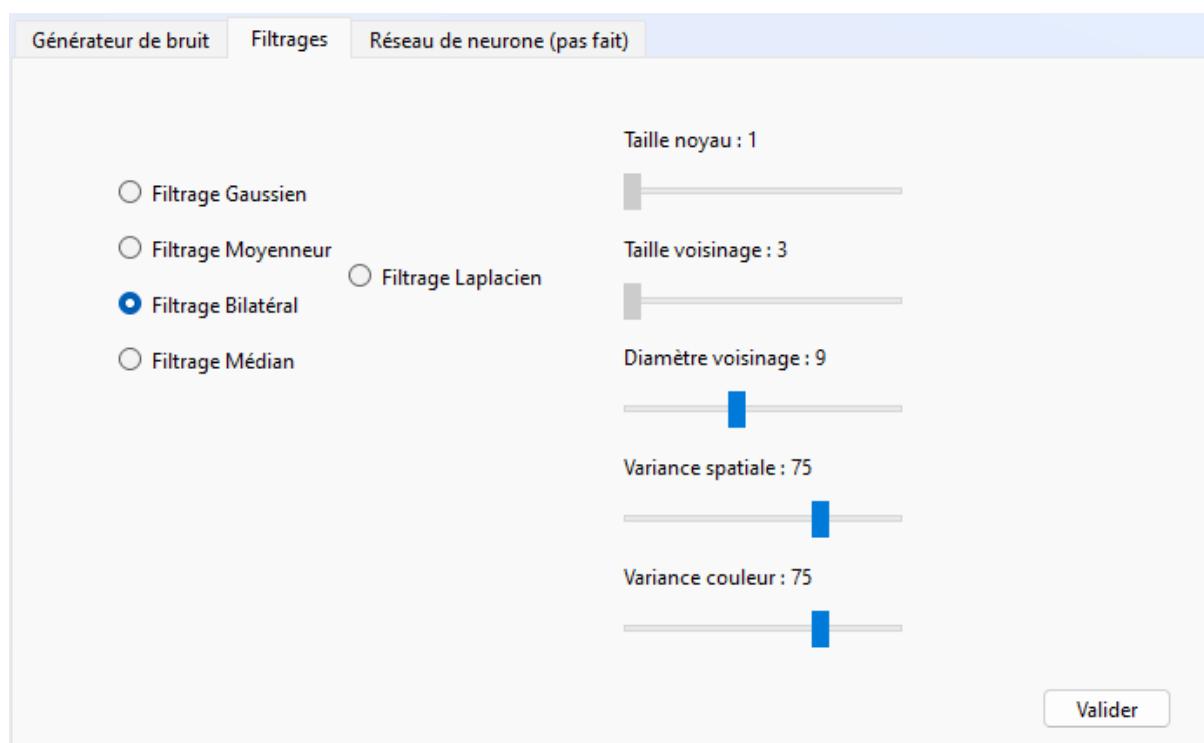
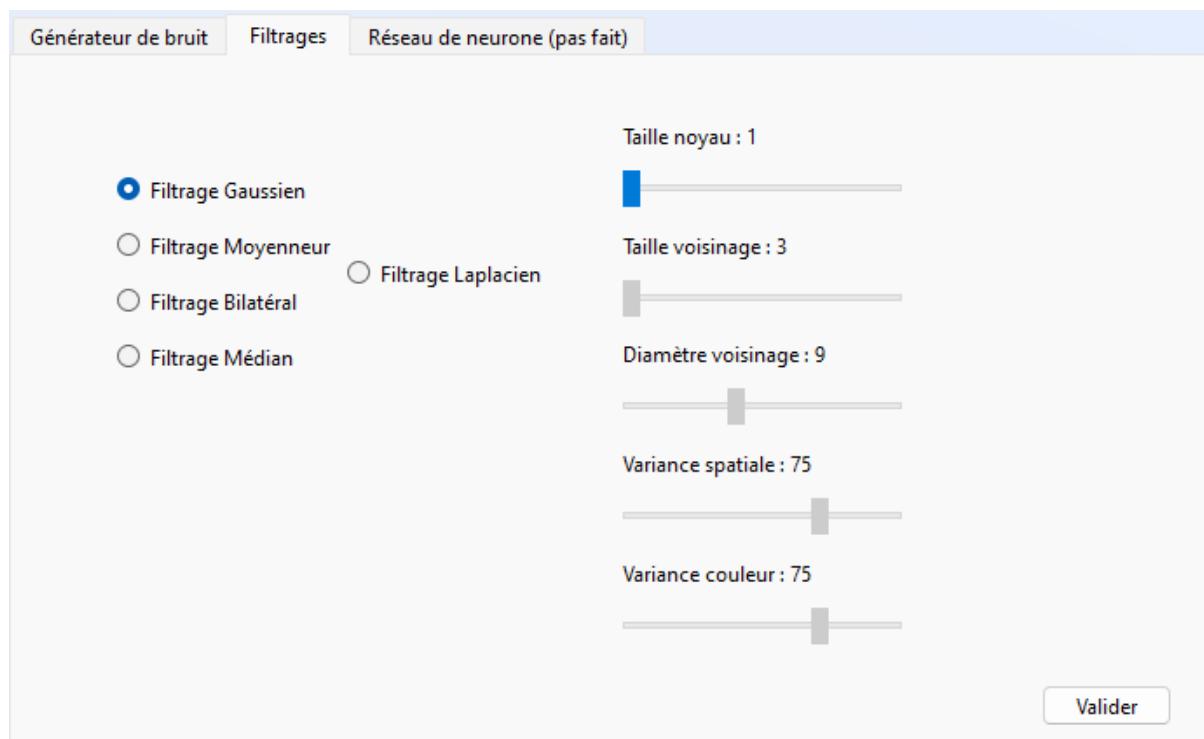
Derrière cette interface se cache plusieurs lignes de code. Celles-ci contiennent notamment l'affichage et la modification des valeurs des sliders. Il faut que ceux-ci soient modifiables seulement si l'image d'origine a été importée. Il a fallu également mettre des valeurs pour les bornes min et max afin de garder des résultats cohérents ainsi que parfois des valeurs de pas. Ci-dessous un exemple pour le slider du radius du filtrage Gaussien :

```
# -- radius
self.ui.slider_radius.setMinimum(1)
self.ui.slider_radius.setValue(self.filter_radius)
self.ui.slider_radius.setMaximum(5)
self.ui.slider_radius.valueChanged.connect(self.update_radius)
self.ui.label_radius.setText(f"Taille noyau : {self.ui.slider_radius.value()}")
```

Également, selon le bouton radio sélectionné, on ne peut modifier que les sliders correspondants au filtre associé.

```
def set_choix_filtre_var(self, choix):
    self.ui.slider_radius.setEnabled(False)
    self.ui.slider_diametre.setEnabled(False)
    self.ui.slider_var_couleur.setEnabled(False)
    self.ui.slider_var_spatiale.setEnabled(False)
    self.ui.slider_taille.setEnabled(False)
    if choix == 1:
        self.ui.slider_radius.setEnabled(True)
    elif choix == 2:
        self.ui.slider_diametre.setEnabled(True)
        self.ui.slider_var_couleur.setEnabled(True)
        self.ui.slider_var_spatiale.setEnabled(True)
    elif choix == 3:
        self.ui.slider_radius.setEnabled(True)
    elif choix == 4:
        self.ui.slider_taille.setEnabled(True)
```

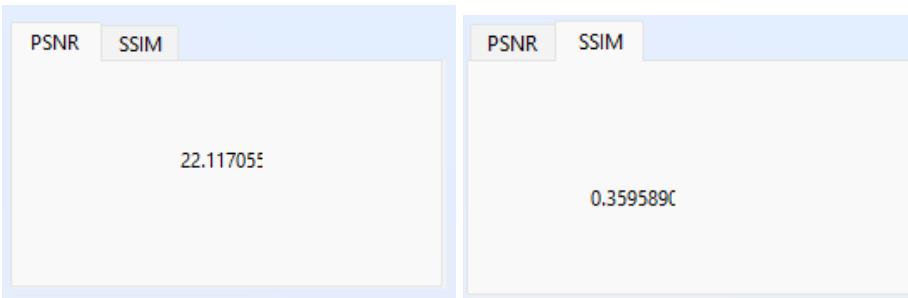
On remarque que le filtrage Laplacien est à part car celui-ci est destiné à améliorer les contours de notre image filtrée. On a fait en sorte que celui-ci ne puisse être activé que si l'image a été modifiée par un filtre.



Afin d'avoir une interaction application/utilisateur agréable, nous avons fait en sorte de ne pas avoir besoin d'appuyer sur le bouton valider lorsque l'on modifie les sliders et l'image modifiée est automatiquement actualisée.

- Mesures :

Nous avons ajouté une partie sur les mesures de qualité entre la photo originale et la photo modifiée.



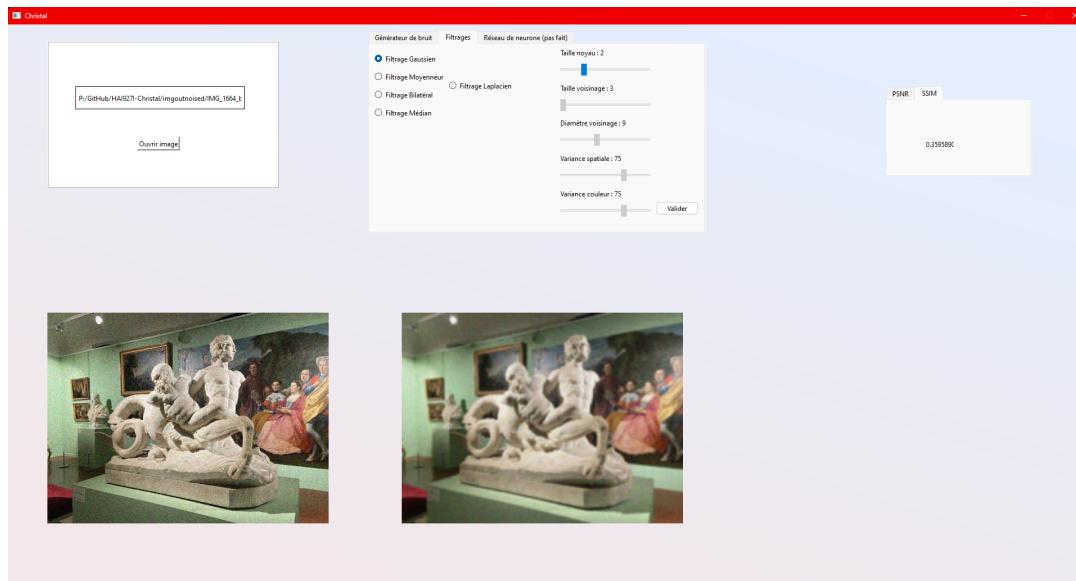
Ci-dessus, l'affichage temporaire des valeurs. En plus du PSNR, et sur les conseils de Nicolas Dibot, nous avons choisi d'ajouter le SSIM (Structural Similarity Index) car cette métrique évalue la similarité structurelle entre deux images. Elle prend en compte la luminance, le contraste et la structure. Cette métrique est considérée comme celle qui offre la meilleure corrélation avec la perception humaine.

Nous ajouterons les éléments manquants tels que l'unité de mesure lors de modifications mineures de l'application.

- Bilan :

Nous pouvons considérer que notre interface graphique est fonctionnelle pour les méthodes traditionnelles et nous pensons qu'il sera facile de l'adapter.

Ci-dessous, l'apparence générale de l'application actuellement.



Implémentation du papier

Pendant cette semaine nous avons essayé d'implémenter le papier *Efficient poisson denoising for photography*.

Le code est composé de plusieurs fonctions, tout d'abord, nous avons une fonction pour la Anscomb transform et une pour son inverse. Pour ces deux fonctions, nous nous sommes juste basés sur les formules trouvées sur internet.

Nous avons ensuite une fonction pour déterminer les zones de l'image sans textures. Pour ce faire, nous nous basons sur ce qui a été proposé dans le avec la formule suivante $T - [J] = \varepsilon w[\nabla[G\sigma * J] > \theta]$ où $G\sigma$ est une convolution gaussienne de variance σ , εw est une érosion morphologique et θ un paramètre d'intensité et pour le gradient de l'image, nous utilisons Sobel. Cependant, les résultats obtenus pour les zones sans textures ne sont pas très convaincants, ce qui va poser un problème pour la suite.

Nous avons ensuite la fonction qui va permettre de calculer le facteur de gain et la densité de photon même si cette dernière ne nous est pas forcément utile. Pour calculer le facteur de gain, ils estiment la moyenne de l'image et la variance du bruit dans un certain nombre de régions, des zones sans textures. Il suffit ensuite de diviser la moyenne par la variance et nous avons notre facteur de gains.

Ensuite, il suffit d'appeler les fonctions dans l'ordre, on va d'abord calculer notre image comportant uniquement les zones sans textures, ensuite, on va pouvoir calculer notre facteur de gain. À partir de là, il suffit de diviser notre image par ce facteur de gain, là, on est donc censé avoir l'image avec le nombre de photons pour chaque pixel. On va donc pouvoir appliquer notre transformée d'Anscombe, notre filtre, on applique ensuite l'inverse de la transformée d'Anscombe et finalement, on multiplie notre image par le facteur de gain pour se retrouver avec notre image finale débruitée.

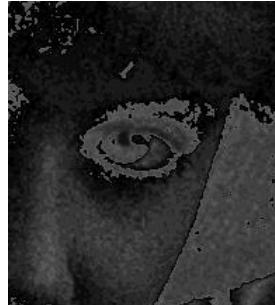
Nous pensons avoir compris le fonctionnement global et les différentes étapes de la méthode présenté, cependant nous avons rencontré quelques difficultés lors de l'implémentation de ce papier.

Tout d'abord, la première difficulté a été l'absence de code ou de pseudo-code mis à disposition, il y a certes toutes les formules présentes, mais il n'est pas toujours évident de traduire ça en code python.

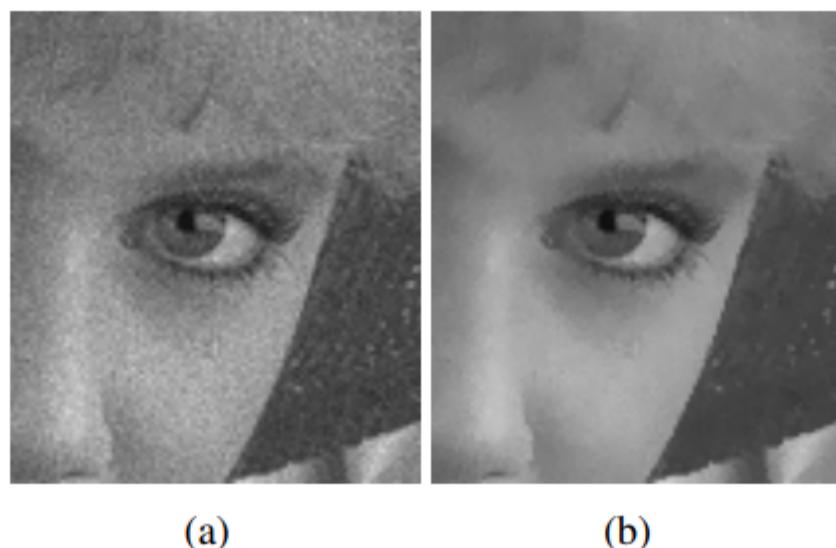
Ensuite, au début, pour le calcul du facteur de gain, nous n'avions pas échantillonnés en régions pour le calcul de la moyenne et de la variance, les premiers résultats n'était pas très convaincants comme vous pouvez le voir avec les images suivantes, celle de gauche est l'image utilisée pour le test et celle de droite le résultat.



Après cette correction et quelques modifications nous nous retrouvons avec ce résultat :



On devine mieux le visage, mais le résultat n'est pas encore celui attendu pour rappel voilà ce qui est obtenu dans le papier



(a)

(b)

Nous nous sommes donc penchés sur les potentielles raisons du problème que nous avons. Le problème vient du calcul du facteur de gain, effectivement avec le facteur de gain que nous obtenions et après avoir appliqué la transformée d'Anscombe, nous devrions nous retrouver avec une variance égale à un et dans notre cas, nous avions une variance qui oscillait entre 6 et 7.

Nous avons donc analysé les différentes étapes en commençant par le calcul des zones sans textures et nous avons pu voir que le résultat de notre masque n'était pas convaincant:



Nous avons donc tout d'abord essayé avec d'autres méthodes pour calculer le gradient d'une image, mais les résultats n'étaient pas meilleurs.

Nous pensons donc qu'il nous faudrait trouver une autre méthode pour calculer ces zones sans textures, la méthode du papier ne fonctionne pas dans notre cas, ils ne précisent également pas comment ils effectuent leur gradient dans leur exemple. Il est aussi possible qu'il y ait eu une erreur dans l'implémentation de leur méthode, mais nous n'avons pas réussi à en trouver la cause.

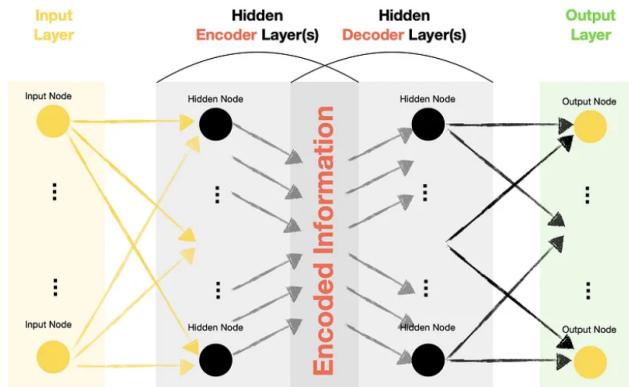
Pour le moment cette tâche va être mise en arrière-plan afin de nous concentrer sur les méthodes avec réseau de neurones, nous continuerons quand même faire quelques recherches en parallèle en nous concentrant sur le problème identifié, mais ce n'est plus la priorité.

Denoising Autoencoder (DAE)

Pour la partie avec réseau de neurones, Nicolas Dibot nous a parlé cette semaine de l'utilisation des Autoencoder pour le débruitage d'images. Souhaitant attaquer la partie réseau de neurones la semaine prochaine, nous nous sommes penchés sur le sujet pour préparer le terrain.

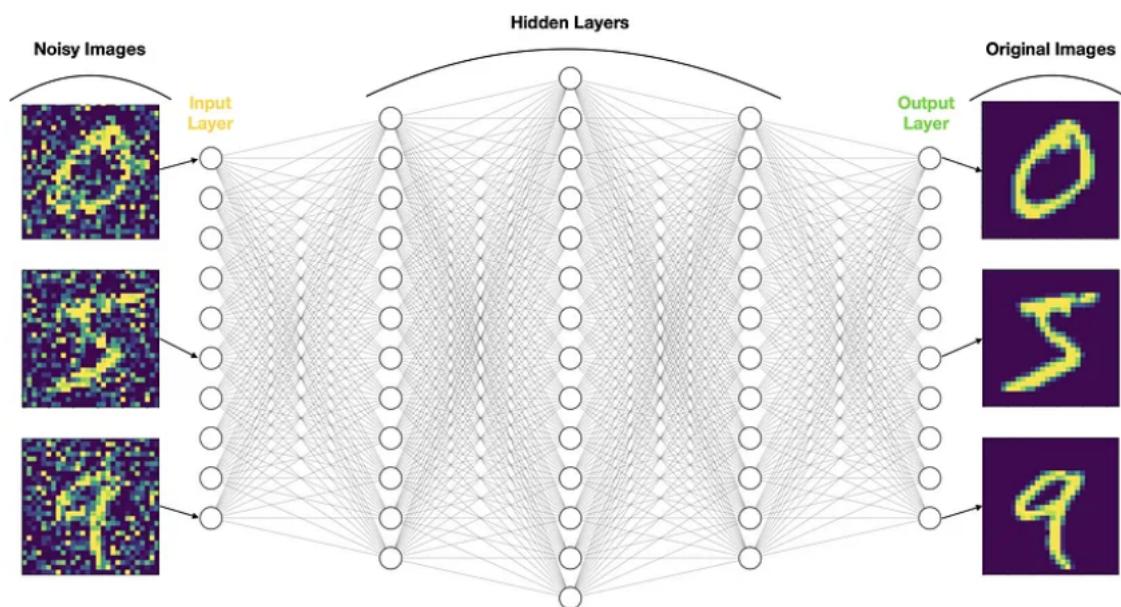
La différence des Autoencoder avec les autres méthodes de réseau de neurones, c'est que les données pour l'entraînement n'ont pas besoin d'être labellisées, ce sont donc des réseaux de neurones non supervisés.

Un DAE va donc être composé d'une couche d'entrée, de couches cachées pour l'encodage et le décodage et d'une de sortie.

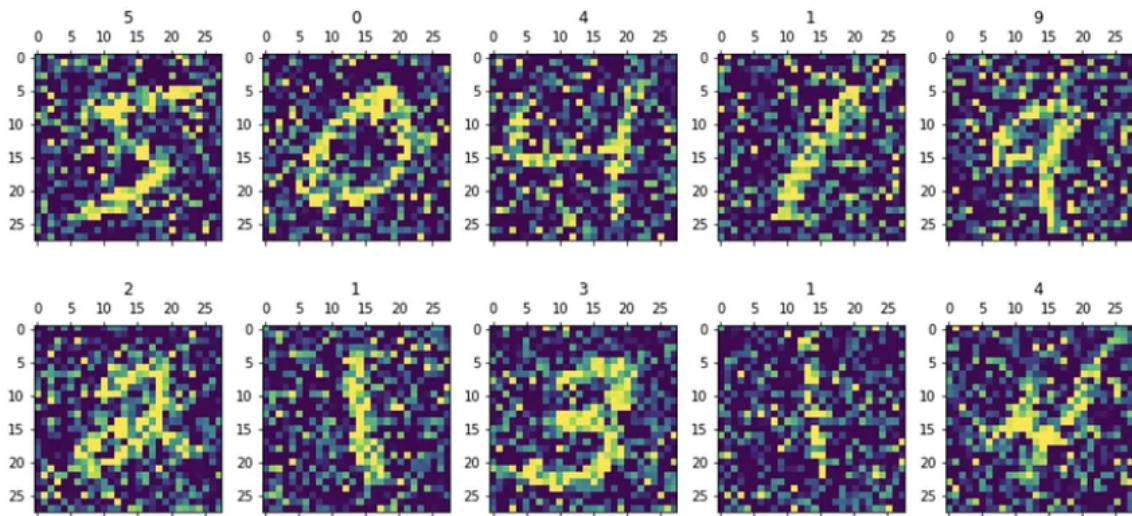


Le but de notre DAE est donc de débruiter un certain type d'images qui va dépendre des données que l'on aura utilisées pour l'entraîner.

Nous avons ensuite suivi le tutoriel proposé sur le site *Denoising Autoencoders (DAE) — How To Use Neural Networks to Clean Up Your Data*. Il propose de mettre en place un DAE simple en utilisant le dataset MNIST. Voici à quoi ressemble l'architecture de notre DAE pour ce tutoriel :



Dans un premier temps, il va charger les données du dataset MNIST et leur rajouter du bruit avec d'avoir un jeu de données avec les images bruitées.

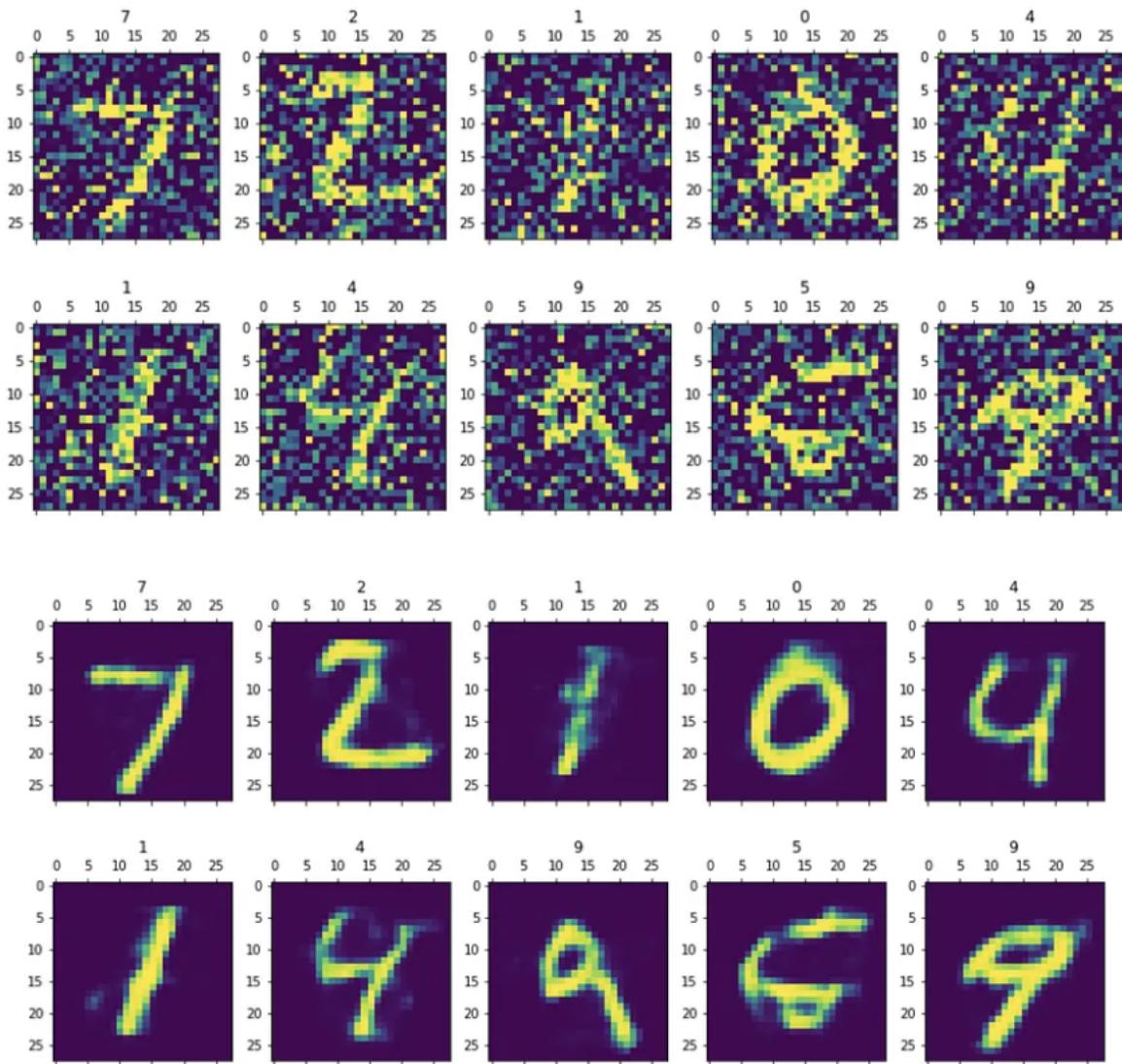


Il va ensuite créer son DAE avec différentes couches, il affiche ensuite l'architecture qui est la suivante :

Model: "Denoising-Autoencoder-Model"

Layer (type)	Output Shape	Param #
<hr/>		
Input-Layer (InputLayer)	[(None, 784)]	0
Encoder-Layer (Dense)	(None, 784)	615440
Encoder-Layer-Normalization (BatchNormalization)	(None, 784)	3136
Encoder-Layer-Activation (LeakyReLU)	(None, 784)	0
Middle-Hidden-Layer (Dense)	(None, 784)	615440
Decoder-Layer (Dense)	(None, 784)	615440
Decoder-Layer-Normalization (BatchNormalization)	(None, 784)	3136
Decoder-Layer-Activation (LeakyReLU)	(None, 784)	0
Output-Layer (Dense)	(None, 784)	615440
<hr/>		
Total params:	2,468,032	
Trainable params:	2,464,896	
Non-trainable params:	3,136	

Maintenant qu'il a son DAE il va pouvoir l'entraîner avec ses images bruitées en entrée et avec comme attente les images débruitées en sortie. Après 20 epochs d'entraînements il se retrouve avec les résultats suivants :



On peut voir que les résultats sont plutôt convaincants, l'auteur précise bien que son modèle n'est pas forcément optimal, ce n'est qu'un point de départ et qu'il sera nécessaire d'expérimenter afin d'avoir un DAE efficace.

Pour le moment, nous avons récupéré ce code pour s'assurer du bon fonctionnement sur nos machines. Nous avons également cherché un dataset pour pouvoir débruiter des photographies, nous avons trouvé celui-ci : *Smartphone Image Denoising Dataset*. Il est composé de plusieurs paires de photos, nous avons donc la version ground truth et la version bruitée.

Notre but maintenant est de remplacer le dataset de MNIST par le nôtre et également de nous pencher plus en détails sur les différents paramètres et les différentes couches que nous pouvons modifier afin d'avoir un résultat optimal. Dans son blog l'auteur met en place un DAE standard mais il est également possible de faire un Convolutionnal DAE, nous avons trouvé un exemple dans la documentation de keras et il pourrait être intéressant de se pencher là-dessus.

Pour la semaine prochaine

Nous allons approfondir nos recherches sur les DAE et adapter notre code au dataset que nous avons trouvé.

Sources

-Efficient poisson denoising for photography: [EFFICIENT POISSON DENOISING FOR PHOTOGRAPHY H. Talbot¹, H. Phelippeau¹, M. Akil¹, S. Bara²](#)

-Denoising Autoencoders (DAE) — How To Use Neural Networks to Clean Up Your Data:<https://towardsdatascience.com/denoising-autoencoders-dae-how-to-use-neural-networks-to-clean-up-your-data-cd9c19bc6915>

-Smartphone Image Denoising Dataset:

<https://www.eecs.yorku.ca/~kamel/sidd/dataset.php>

-Convolutional autoencoder for image denoising:

<https://keras.io/examples/vision/autoencoder/>