

Chapitre 1

Acquisition des images par marquage des spins artériels

1.1 Principe du marquage des spins artériels

L'approche ASL est une technique d'imagerie par résonance magnétique au cours de laquelle deux types d'acquisitions des coupes sont réalisées : une acquisition qui se base sur le marquage magnétique des protons du sang comme traceur endogène. Ce marquage est effectué par une impulsion de radiofréquence en amont du volume d'intérêt. Après l'impulsion de marquage, une acquisition du volume d'intérêt est effectuée après un délai post-marquage, elle permet d'obtenir les images marquées. D'une autre part, une acquisition de contrôle de même volume d'intérêt mais sans marquage est effectuée en produisant les images de contrôles. Pour obtenir une cartographie de perfusion, une soustraction entre les images marquées et les images de contrôles est nécessaire, il faut répéter cette soustraction plusieurs fois (en général, 30 à 60 paires marqué-contrôle) puisque la différence du signal entre les deux acquisitions est très faible.

La figure suivante représente les principales étapes de la technique ASL décrites dans l'article de Marjorie Villien (2012) [1] :

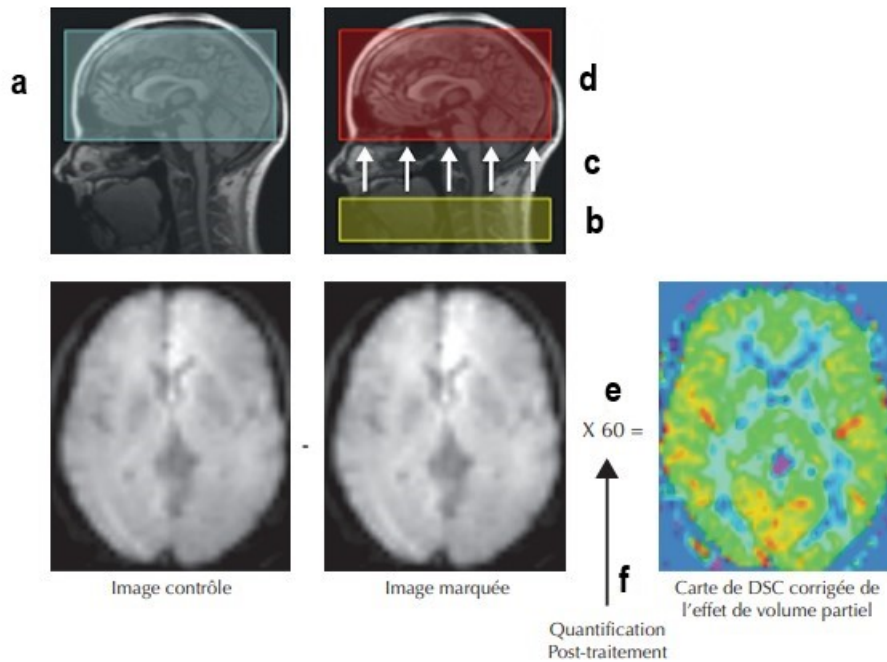


FIGURE 1.1 – Processus de la technique ASL.

1. Acquisition du volume d'intérêt sans marquage préalable des protons du sang artériel d'amont : il est réalisé le plus souvent par une impulsion d'inversion. De plus, les spins présents dans la coupe de marquage sont inversés et sont transportés avec le sang vers la région d'intérêt pour obtenir une image contrôle (voir la figure 1.1(a)).
2. Marquage des protons du sang artériel d'amont par une impulsion de radiofréquence (voir la figure 1.1(b)).
3. Attente d'un délai entre le marquage et l'arrivée des protons marqués au niveau du volume d'intérêt : les spins inversés reviennent à leur aimantation d'équilibre en suivant la constante de temps d'inversion (voir la figure 1.1(c)).
4. Acquisition du volume d'intérêt : après l'impulsion de marquage, une acquisition d'une image des tissus est obtenue avec une technique d'imagerie rapide sur la région d'intérêt. Lors de cette acquisition, le signal porte l'aimantation des spins statiques de la région d'intérêt à laquelle s'ajoute l'aimantation amenée par le pool sanguin marqué dans le volume exploré pour obtenir une image marquée (voir la figure 1.1 (d)).

5. Soustraction image contrôle-image marquée et obtention d’une image pondérée en perfusion avec un nombre de répétitions qui est en général de 30 à 60 : pour obtenir le signal des spins marqués qui ont perfusé les tissus de la région d’intérêt, on fait une soustraction entre l’intensité du signal dans les images de contrôle et celle dans les images de marquage. En effet, le rapport obtenu est très faible, d’où la nécessité de répéter les images afin d’obtenir un rapport signal à bruit suffisant (voir la figure 1.1(e)).
6. Application d’un modèle de quantification et post-traitement qui permet d’obtenir une cartographie quantitative du débit sanguin cérébral : la qualité des cartographies de ce DSC doit être améliorée avec un traitement nécessaire sur les images qui est centré sur la correction des mouvements, un recalage morphologique sur l’image ainsi que la correction des effets de volume (voir la figure 1.1(f)).

1.2 Techniques principales de marquage d’ASL

D’après l’article de Clément Debacker (2014) [2], il existe trois types principaux de marquage d’ASL : l’ASL pulsé (PASL), l’ASL continu (CASL) et l’ASL pseudo-continu (pCASL).

1.2.1 Méthode ASL continu

1.2.1.1 Principe de la méthode CASL

La technique ASL continu utilise une impulsion longue de plusieurs secondes d’inversion induite par le flux, appliquée en continu pendant tout le marquage pour réaliser l’inversion du sang. Cette impulsion est beaucoup plus longue que les impulsions classiques qui sont de l’ordre de plusieurs ms, ce qui fait que de nombreux systèmes IRM ne sont pas capables d’appliquer ce type d’impulsion. Le tableau suivant résume les avantages et les inconvénients de la méthode CASL :

Avantages	Inconvénients
Elle produit un meilleur rapport signal sur bruit ; le sang marqué arrive plus rapidement à la zone imagée	Il ne permet pas de réaliser une acquisition multi-coupes, nécessite une antenne de marquage spécifique.
	Il n'est pas toujours possible de réaliser une impulsion continue de plusieurs secondes.
	Un coût supplémentaire et un tel matériel n'est pas toujours disponible.
	Elle génère un dépôt d'énergie élevé dans les tissus, souvent supérieur aux limites acceptables à 3 ou 7 Tesla.

TABLE 1.1 – Avantages et Inconvénients de CASL

1.2.2 Méthode ASL pulsé

1.2.2.1 Principe de la méthode PASL

La technique PASL utilise de brèves impulsions de radiofréquences en se basant sur l'acquisition du signal après un délai TI entre l'image contrôle et l'utilisation d'une radiofréquence courte de marquage. Le tableau suivant résume les avantages et les inconvénients de la méthode PASL :

Avantages	Inconvénients
Elle limite le dépôt d'énergie dans les tissus et les effets de transfert d'aimantation.	Signal de perfusion est plus faible.
Impulsions courtes plus faciles à mettre en œuvre (généralement 10-15 ms) qui inversent les spins dans une région spécifique.	

TABLE 1.2 – Avantages et Inconvénients de PASL

1.2.3 Méthode ASL pseudo-continu

1.2.3.1 Principe de la méthode pCASL

Cette méthode repose sur le même principe d'inversion adiabatique induite par le flux que la méthode CASL. De plus, on mime une inversion continue par de très brèves impulsions RF dont la durée est comprise entre 200 et 500 μs , répétées toutes les 400 à 1500 μs . Le tableau suivant résume les avantages et les inconvénients de la méthode pCASL :

Avantages	Inconvénients
Elle combine les avantages des deux autres techniques CASL et PASL avec un rapprochement au CASL.	Cette méthode présente le désavantage d'être plus sensible aux inhomogénéités de champ magnétique dans le plan de marquage que les autres méthodes.
Elle utilise des impulsions de radiofréquence avec une courte durée, d'où la conservation d'un bon signal sur bruit avec une moins sensibilité au temps de transit.	
Elle est pratique à utiliser en routine sur une IRM de type clinique que le CASL.	

TABLE 1.3 – Avantages et Inconvénients de pCASL

1.3 Conclusion

D'après l'analyse de chaque méthode de la technique ASL, on conclut que la méthode PASL est la plus avantageuse, toutefois cette méthode produit des images bruitées à cause de la faiblesse du signal de perfusion, ce qui nous mène à l'application des algorithmes de débruitage pour essayer de réduire le bruit ces images.

Chapitre 2

Méthodes de débruitage

2.1 Introduction

Dans le chapitre précédent, on a vu que l’ASL pulsé, un type de la technique ASL produit un bruit cause de faiblesse de son signal. Pour ce faire, on a consacré ce chapitre pour analyser trois algorithmes de débruitage afin de comparer leur performance au niveau de la réduction du bruit des images d’une séquence ASL. Les deux premiers algorithmes de l’apprentissage profond sont respectivement : autoencodeur débruiteur (DAE) et le réseau neuronal convolutif de débruitage (DnCNN). Le troisième est un algorithme standard qui est le filtrage non local moyen (NLmeans).

2.2 L’apprentissage profond

Le domaine du traitement de l’image en intelligence artificielle est composé de plusieurs branches. Dans cette partie, on va aborder le réseau neuronal convolutif pour le débruitage d’images médicales de type IRM en niveaux de gris.

2.2.1 Définition de l’approche

L’apprentissage profond «Deep Learning» est un sous-domaine de l’intelligence artificielle (voir la figure 2.1). Ce terme désigne l’ensemble des techniques d’apprentissage automatique, il s’agit d’une forme d’apprentissage fondée sur des approches mathématiques utilisées pour modéliser des données.

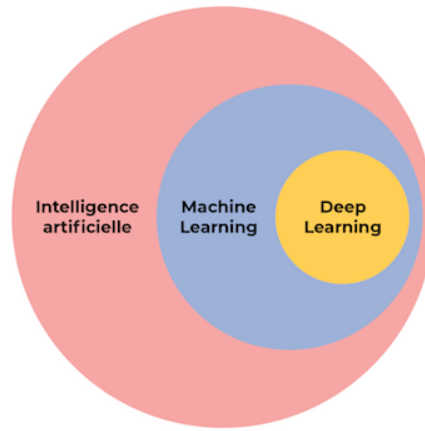


FIGURE 2.1 – Branches de l'intelligence artificielle.

2.2.2 Réseau neuronal artificiel

L'apprentissage profond est inspiré par les réseaux de neurones biologiques constituant un cerveau humain. Ces neurones sont interconnectés afin de traiter et analyser l'information, ainsi de la mémoriser, aussi ils sont capables de résoudre le problème de la plus meilleure façon possible et de le comparer par des situations passées (voir la figure 2.2). De plus, un réseau neuronal artificiel est composé de plusieurs couches de neurones, chacune de ces couches permet de recevoir et d'interpréter les informations de la couche précédente, ce réseau est capable d'accepter un signal en entrée, de le modifier et de le restituer à une ou plusieurs sorties, de différentes manières.

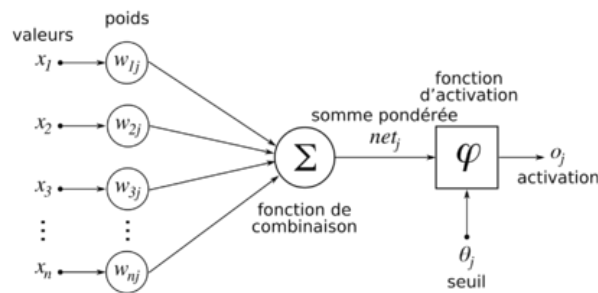


FIGURE 2.2 – Architecture d'un réseau neuronal artificiel

La figure 2.2 représente le principe de fonctionnement d'un réseau de neurones artificiels dans lequel on associe un poids (w_{ij}) à chaque valeur (x_i), puis une fonction de combinaison est appliquée pour réduire le nombre des paramètres du réseau, cette fonction sert à résumer l'information de plusieurs neurones voisins en une seule information moyenne, après une fonction d'activation est

appliquée à cette somme pondérée afin de faciliter l'extraction des caractéristiques complexes d'une image.

2.2.3 Réseau neuronal convolutif

Dans le domaine de l'apprentissage profond, le réseau neuronal convolutif «Convolutional Neural Network» (CNN) désigne une sous-catégorie de réseaux de neurones, il présente donc toutes les caractéristiques listées ci-dessus. Cependant, il est spécialement conçu pour traiter des images en entrée.

2.2.4 Architecture du réseau neuronal convolutif

L'architecture du réseau neuronal convolutif dispose en amont d'une succession de blocs de traitement qui se compose d'une à plusieurs couches bien distinctes : couches de Convolution (CONV), couches de pooling (POOL), couches de Correction (ReLU), couches entièrement-connectée «Fully-connected» (FC) et couches de perte (LOSS).

2.2.5 Couches de réseaux de neurones convolutifs

2.2.5.1 Couches de convolution

Un réseau neuronal convolutif peut être constitué d'une ou plusieurs couches de convolution, dont la première couche apprendra les petits motifs, la deuxième apprendra des motifs plus importants constitué des caractéristiques des premières couches. Dans cette couche, l'image est découpée en petites zones, ce sont des fenêtres de convolution en construisant des matrices avec des valeurs et en appliquant un filtre qui est égale au nombre d'opérations à réaliser, puis il vient la multiplication de chaque matrice avec un kernel en utilisant un pas «strides» qui représente le nombre de saut à faire pour déplacer notre fenêtre de convolution, jusqu'à ce qu'on termine. Enfin, on obtient des cartes de caractéristiques, en effet le nombre de ces cartes dépend aux nombres de filtres appliqués à l'image (voir la figure [2.3](#)).

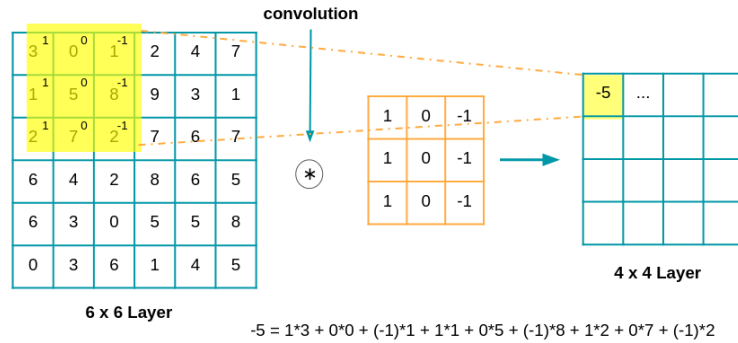


FIGURE 2.3 – Processus de fonctionnement de la couche de convolution.

Cette figure représente le processus de fonctionnement de la couche de convolution d'une image d'entrée de taille 6x6 puis on extrait les motifs ou kernel de 3x3 pixels, après il vient la multiplication de ce motif par une matrice de convolution en ajoutant un pas de 1x1 pour passer sur chaque pixel de l'image, enfin la génération des cartes de caractéristiques de sorties «feature-maps» de taille 4x4 égales aux nombres de filtres appliqués à l'image.

2.2.5.2 Couches de pooling

Après l'obtention des cartes de caractéristiques dans la couche de convolution, la couche de pooling sert à réduire leurs tailles (par exemple avec 64 filtres on obtient 64 cartes de caractéristiques). Cette opération de pooling sert à extraire la moyenne des poids associés à chaque pixel, mais il est plus efficace d'utiliser le maximum que la moyenne, puisque le max pooling sert à extraire seulement la valeur maximale, c-à-d la valeur la plus importante de chaque motifs des cartes de caractéristiques (voir la figure 2.4).

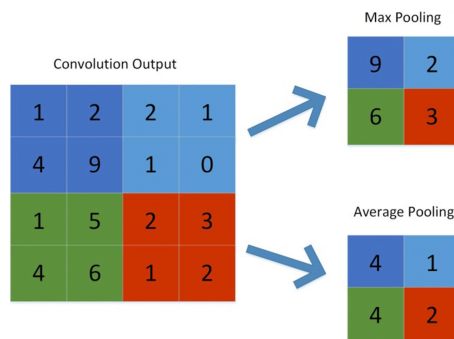


FIGURE 2.4 – Processus de fonctionnement de la couche de pooling et la différence entre le max pooling et l'average pooling.

Dans cette figure, on remarque qu'après la convolution, si un average pooling est appliqué, la moyenne des valeurs est prise en compte, par contre si un max pooling est appliqué seulement le maximum des valeurs qui est pris en compte, ce qui permet d'extraire uniquement les données importantes.

2.2.5.3 Couches de correction

Cette couche sert à appliquer une fonction d'activation qui est une fonction non-linéaire aux cartes de caractéristiques obtenues dans la couche de convolution afin de rendre les caractéristiques complexes où la fonction linéaire ne fonctionne pas à des données non-linéaires. Cette couche sert à remplacer toutes les valeurs négatives reçues en entrée par des zéros.

2.2.5.4 Couches entièrement-connectée «Fully-connected» (FC)

Après les couches de convolution et de max pooling, il vient la couche FC. Cette couche est très importante dans un réseau de neurone car elle est entièrement connectée puisque tous les neurones de l'entrée de la couche sont connectés avec celle de la sortie, ce qui signifie qu'ils ont l'accès à la totalité des informations de l'entrée (voir la figure 2.5).

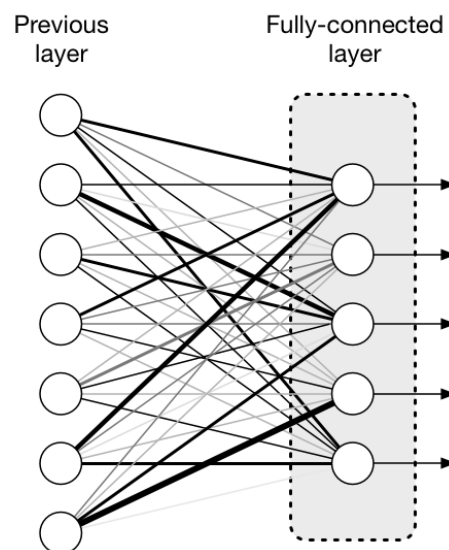


FIGURE 2.5 – Schéma d'une couche FC où chaque neurone de la couche est connecté à tous les neurones de la couche précédente.

2.2.5.5 Couches de perte

La couche de perte est la dernière couche d'un réseau de neurone. Elle sert à calculer l'erreur entre le signal prévu et réel. Dans cette couche, il y a des diverses fonctions de perte adaptées à différentes tâches peuvent y être utilisées.

2.3 Autoencodeur débruiteur

2.3.1 Autoencodeur

L'autoencodeur est un algorithme de compression de données, où on ne peut compresser que des données similaires à celles sur lesquelles il a été formé. Avec un autoencodeur, les données décompressées en sortie seront dégradées par rapport aux entrées originales. En effet, il est composée de 3 parties élémentaires (voir la figure 2.6) :

Fonction d'encodage.

Fonction de décodage.

Fonction de perte, fonction de distance entre la quantité de perte d'information entre les données compressées et les données décompressées.

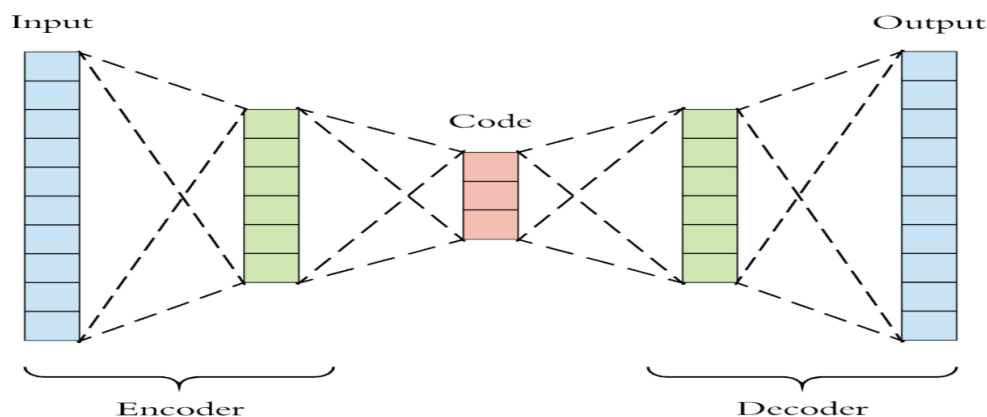


FIGURE 2.6 – Architecture de l'autoencodeur.

Cette figure modélise l'architecture d'un autoencodeur décrit dans le dépôt en github «Combined Denoising and Suppression of Transient Artifacts in Arterial Spin Labeling MRI Using Deep Learning» [3]. En effet, cet algorithme nécessite une donnée d'entrée qui sera compressée par la suite dans la phase encodeur, juste après il vient la phase de la décompression en conservant les dimensions réelles aussi fidèlement que possible.

2.3.2 Modèle de DAE

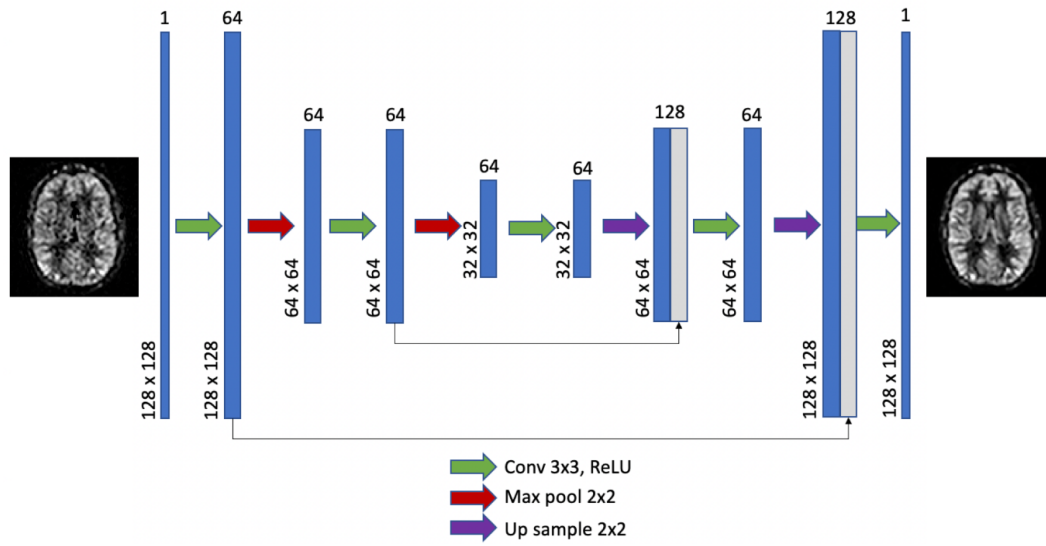


FIGURE 2.7 – Architecture du modèle l’autoencodeur Débruiteur (DAE).

La figure 2.7 représente l’architecture du modèle de l’autoencodeur qu’on va appliquer par la suite pour le débruitage des images d’une séquence ASL de type IRM, ce modèle contient deux composants principaux :

Codeur : comme premier étape il faut mettre une image en entrée de dimension 128x128, ici dans ce modèle, il y a trois couches de convolution avec un kernel de 3x3 et 64 couches de filtres, chaque couche est suivie par des fonctions d’activation ReLU, cela va donner 64 cartes de caractéristiques de même dimensions que l’image d’entrée, il ya aussi deux couches de max pooling 2x2 qui sont appliquées après chaque convolution pour extraire seulement la valeur la plus importante de chaque motif des cartes de caractéristiques, ce qui conduit à la phase codeur de générer 64 cartes de caractéristiques de dimension 32x32 (voir la figure 2.7).

Décodeur : où un Upsampling 2x2 est appliqué après chaque fonction de convolution, le Upsampling rend l’image d’entrée à ses dimensions réelles en améliorant sa qualité, il vient ensuite la convolution mais cette fois sans la fonction d’activation ReLU et avec une seule couche de filtre afin d’obtenir l’image finale débruitée (voir la figure 2.7).

2.4 Réseau de neurones convolutifs de débruitage

2.4.1 Principe de réseau de neurones convolutifs de débruitage

Le but du débruitage d'image est de récupérer une image propre x à partir d'une observation bruyante y qui suit un modèle de dégradation de l'image $y = x + v$. Une hypothèse commune est que v est un bruit blanc gaussien additif avec une écart-type σ . Le réseau de neurones convolutif de débruitage DnCNN est conçu pour prédire l'image résiduelle \hat{v} , c'est-à-dire la différence entre l'observation bruyante et l'image propre latente, plutôt que de sortir directement l'image débruitée \hat{x} . En d'autres termes, le DnCNN supprime implicitement l'image propre latente avec les opérations dans les couches cachées.

2.4.2 Architecture du réseau de neurones convolutifs de débruitage

Le réseau DnCNN permet de prédire l'image résiduelle qui représente la différence entre l'observation bruitée et l'image propre. Pour un meilleur compromis entre la performance et l'efficacité, la taille des filtres convolutifs est fixée à 3×3 et toutes les couches de mise en commun sont supprimées. Il a été souligné que la taille du champ réceptif du DnCNN avec une profondeur de d est corrélée à la profondeur de l'image et donc devrait être $(2d+1)(2d+1)$. Pour un débruitage gaussien avec un certain niveau de bruit, la taille du champ réceptif de DnCNN est fixée à 35×35 avec une profondeur correspondante de 17. Pour le DnCNN, on adopte la formulation de l'apprentissage résiduel afin d'entraîner une fonction de mappage résiduelle $R(y)$ pour former une cartographie résiduelle $R(y) \approx v$ et on a alors $x = y - R(y)$. En effet, il y a trois types de couches (voir la figure 2.8).

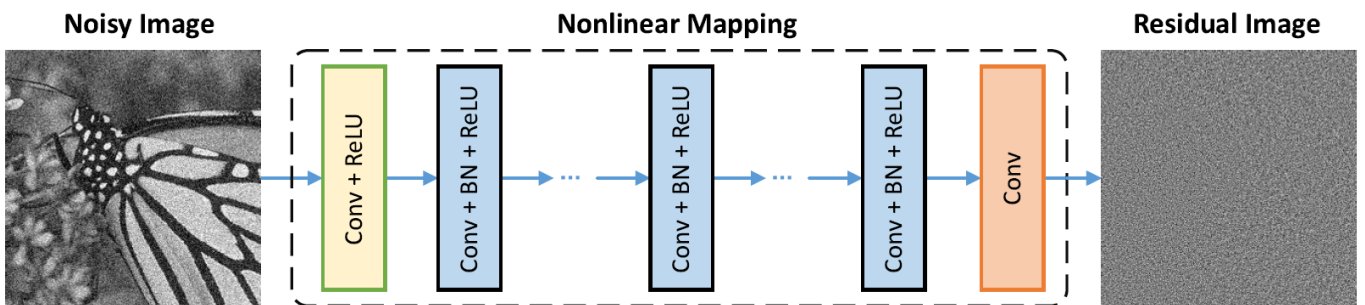


FIGURE 2.8 – Architecture du réseau de neurones convolutifs de débruitage.

Conv+ReLU : pour la première couche, 64 filtres de taille 3×3 sont utilisés pour générer 64

cartes de caractéristiques. En effet, en incorporant la convolution avec ReLU, le DnCNN peut progressivement séparer la structure de l'image de l'observation bruyante à travers les couches cachées.

Conv+BN+ReLU : pour les couches 2 à (d-1), 64 filtres de taille $3 \times 3 \times 64$ sont utilisés, et une normalisation par lot «batch normalization» est ajoutée entre la convolution et ReLU.

Convolution : pour la dernière couche, c filtres de taille $3 \times 3 \times 64$ sont utilisés pour reconstruire la sortie.

2.5 Filtrage non local moyen

Le filtrage non local moyen [6] est un algorithme qui sert à traiter les images pour les débruiter. Ce filtrage prend la moyenne de tous les pixels de l'image, pondérée par la similarité de ces pixels dans un voisinage ayant comme centre le pixel cible. En effet, le filtrage non local donne une clarté post-filtrage beaucoup plus grande et une perte de détails moindre dans l'image par rapport aux algorithmes de moyenne local qui prennent la valeur moyenne d'un groupe de pixels qui entourent un pixel cible pour lisser l'image. L'algorithme de NLmeans est défini par la formule suivante :

$$NLu(p) = \frac{1}{C(p)} \int f(d(B(p), B(q))u(q) dq$$

où $d(B(p), B(q))$ est la distance euclidienne entre les plages d'image centrées respectivement sur les points p et q, f est une fonction décroissante et C(p) est un facteur de normalisation. q, f est une fonction décroissante et C(p) est le facteur de normalisation.

Lors du calcul de la distance euclidienne $d(B(p), B(q))$, tous les pixels de la parcelle B(p) ont la même importance et donc le poids $f(d(B(p), B(q)))$ peut être utilisé pour débruiter tous les pixels du bloc B(p) et pas seulement p.

2.5.1 Les paramètres de l'algorithme

Le paramètre σ joue un rôle très important dans cet algorithme, car il sert à déterminer la taille du bloc voisinage du filtrage. Il prend des valeurs entre 0 et 100, ce paramètre affecte la taille du bloc car si sa valeur augmente on a besoin d'un bloc plus grand pour que la comparaison de blocs soit suffisamment robuste (voir la figure 2.9).

σ	Gray		h
	Comp. Patch	Res. Block	
$0 < \sigma \leq 15$	3×3	21×21	0.40σ
$15 < \sigma \leq 30$	5×5	21×21	0.40σ
$30 < \sigma \leq 45$	7×7	35×35	0.35σ
$45 < \sigma \leq 75$	9×9	35×35	0.35σ
$75 < \sigma \leq 100$	11×11	35×35	0.30σ

FIGURE 2.9 – Paramètres de l’algorithme sur une image grise.

D’après le tableau, on remarque que pour chaque intervalle de valeurs de σ appliquées sur les images grises, correspond une fenêtre de bloc ainsi que le paramètre de filtrage h défini par $h = k\sigma$ (k diminue en augmentant σ). Lorsque σ augmente, la fenêtre du bloc augmente.

2.6 Conclusion

Après l’analyse des trois algorithmes, on conclut que les réseaux de neurones convolutifs peut se révéler plus efficaces face à la méthode standard le filtrage non local moyen.

Chapitre 3

Application et comparaison des trois algorithmes de débruitage DAE, NLmeans et DnCNN

3.1 Introduction

Dans le chapitre précédent, on a décrit les trois algorithmes de débruitage : autoencodeur débruiteur, filtrage non local moyen et réseau neuronal convolutif de débruitage. Dans ce chapitre, on va tester ces algorithmes sur des images bruitées d’une séquence ASL du type IRM.

3.2 Données d’entrée utilisées

On a utilisé des données citées dans le dépôt en github [4] qui sont des séquences PASL contenant des images brutes, ce fichier «dMRaw.nii.gz» de type nifti est visualisé à l’aide d’un logiciel appelé ImageJ [7] (voir la figure 3.1) pour l’analyse des images. Il sert à afficher, éditer, analyser, traiter, sauvegarder et imprimer des images de 8, 16 et 32 bits et supporte la plupart des principaux formats dont «TIFF, GIF, JPEG, BMP, DICOM, FITS et raw».

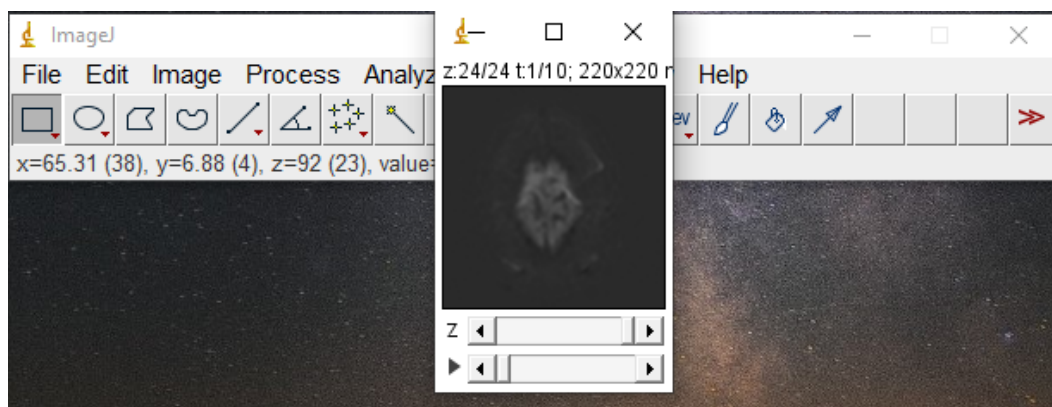


FIGURE 3.1 – Visualisation des données à l'aide du logiciel ImageJ.

Dans notre cas, on a utilisé dix images d'une séquence de ces données (voir la figure 3.2).

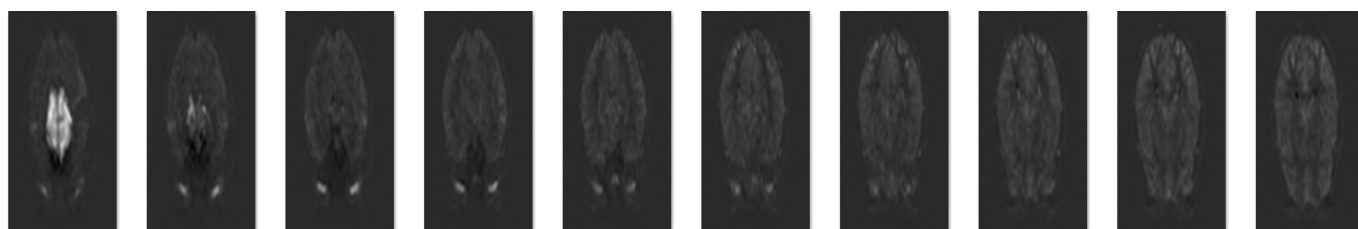


FIGURE 3.2 – Les images bruitées d'ASL utilisées.

3.3 Création du modèle autoencodeur débruiteur DAE

On a utilisé le code source hébergé dans un dépôt en github «Combined Denoising and Suppression of Transient Artifacts in Arterial Spin Labeling MRI Using Deep Learning» [4] qui contient le code source nécessaire à la création du modèle DAE décrit.



3.3.1 L'environnement de développement utilisé

Pour les deux premiers algorithmes basés sur le «Deep Learning», on a utilisé le langage de programmation Python car il répond parfaitement aux besoins de l'apprentissage profond.



D'une autre part, on a utilisé l'environnement de développement «Google Colaboratory», adapté au «deep Learning» qui permet d'écrire et d'exécuter le code Python par le biais du navigateur.



3.3.2 Les modules utilisés

Os : destiné à interagir avec le système d'exploitation (voir la figure 3.3).

Numpy : destiné à manipuler des matrices ou tableaux multidimensionnels ainsi que des fonctions mathématiques opérant sur ces tableaux (Voir la figure 3.3).

Matplotlib : destiné à tracer et visualiser des données sous forme de graphiques (voir la figure 3.3).

NiBabel : fournit un accès en lecture +/- écriture à certains formats de fichiers médicaux et de neuro-imagerie courants. NiBabel est le successeur de PyNifti (Voir la figure 3.3).

Keras : interface de programmation d'application (API) de réseau neuronal pour Python, utilisée pour construire des modèles d'apprentissage automatique (Voir la figure 3.3).

Scikit-image : conçu pour le prétraitement des images (Voir la figure 3.3).

Pickle : permet de sauvegarder dans un fichier, au format binaire, n'importe quel objet Python (Voir la figure 3.3).

```
import numpy as np
from matplotlib import pyplot as plt
import nibabel as nib
import os
from keras.models import load_model
from skimage.transform import resize
import pickle
from keras.layers import Input, Conv2D, MaxPooling2D, UpSampling2D, BatchNormalization, Activation, Dropout
from keras.layers.merge import concatenate
from keras.models import Model
```

FIGURE 3.3 – Librairies utilisées pour le modèle DAE.

3.3.3 Description des principales fonctionnalités du code source

Le code source est composé d'une classe Dae utilisée pour implémenter le modèle de «Denoising Autoencoder» entraîné sur des données d'entrée soit de 2D, 3D ou 4D qui comporte quatres méthodes principaux qui sont :

`_init_(self, dMRaw=None, dMRawFile=None)` : ce constructeur permet de charger le fichier «DaeTrainedModel.h5», le fichier «DaeTrainedModelMetaData.pkl» et permet aussi de charger des données d'entrée. Pour notre cas, on a utilisé des données d'entrée : «dMraw.nii.gz» ayant comme dimension 2D.

`ApplyModel(self)` : cette méthode permet d'appliquer le modèle DAE sur les données d'entrée en définissant les zones d'intérêt qui contiennent le bruit pour générer les données de sortie débruitées.

`ShowSlice(slice=None, rep=None)` : cette méthode permet de visualiser une paire d'image originale bruitée et image débruitée en spécifiant le numéro de la tranche à afficher, par défaut c'est la tranche centrale et en spécifiant aussi le nombre de répétitions qui est par défaut 1.

`ShowTraining(self)` : cette méthode permet d'afficher un tracé de la perte d'entraînement et de validation pendant l'entraînement du modèle.

D'une autre part, il y a la partie de création du modèle DAE qui se compose de :

`Conv2D_Block` qui sert à appliquer la convolution aux données d'entrées en spécifiant le nombre de filtres, la fonction d'activation, le padding ainsi que le kernel.

Dans notre cas :

`n_filters = 64.`

`Activation = 'relu'.`

`Kernel_size = 3.`

`Padding = 'same'` qui signifie que l'image en sortie doit avoir les mêmes dimensions que l'image d'entrée.

Autoencodeur qui retourne le modèle d'architecture qui est composée de deux parties :

Codeur : permet d'appliquer deux couches de convolution en utilisant la méthode `Conv2D_Block()`. Chacune des couches est suivi d'une couche maxpooling 2x2 pour réduire les dimensions des données.

Décodeur : permet d'appliquer deux couches de convolution en utilisant `Conv2D_Block()`.

Chacune des couches est suivi d'une couche Upsampling 2x2 pour rendre les données à ses dimensions initiales. Ensuite, il vient la dernière couche de convolution qui est appliquée sans fonction d'activation avec une seule couche de filtre afin d'obtenir les données finales.

3.3.4 Application du code source

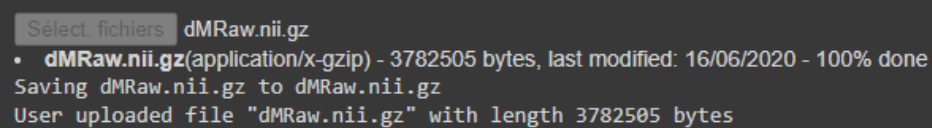
Dans cette partie, on décrit l'application du code source du DAE [4] étape par étape :

Téléchargement des données vers google colaboratory (voir la figure 3.4).

```
from google.colab import files

uploaded = files.upload()

for fn in uploaded.keys():
    print('User uploaded file "{name}" with length {length} bytes'.format(
        name=fn, length=len(uploaded[fn])))
```



Sélectionner fichiers dMRaw.nii.gz

- dMRaw.nii.gz(application/x-gzip) - 3782505 bytes, last modified: 16/06/2020 - 100% done

Saving dMRaw.nii.gz to dMRaw.nii.gz

User uploaded file "dMRaw.nii.gz" with length 3782505 bytes

FIGURE 3.4 – Téléchargement des données d'entrée pour le modèle DAE

Importation du fichier `DaeTrainedModel.h5` qui contient le modèle DAE entraîné et le fichier «`DaeTrainedModelMetaData.pkl`» qui contient les métadonnées pour le modèle entraîné (voir la figure 3.5).

```

from google.colab import files

uploaded = files.upload()

for fn in uploaded.keys():
    print('User uploaded file "{name}" with length {length} bytes'.format(
        name=fn, length=len(uploaded[fn])))

```

Sélect fichiers Aucun fichier choisi Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

Saving DaeTrainedModelMetaData.pkl to DaeTrainedModelMetaData.pkl
 Saving DaeTrainedModel.h5 to DaeTrainedModel.h5
 User uploaded file "DaeTrainedModelMetaData.pkl" with length 1786 bytes
 User uploaded file "DaeTrainedModel.h5" with length 1237688 bytes

FIGURE 3.5 – Téléchargement du modèle DAE entraîné et ces métadonnées.

Application de la méthode `applyModel()` (voir la figure 3.6) :

```

dae=Dae()
dae.applyModel()

```

FIGURE 3.6 – Méthode `applyModel()` de la classe DAE.

Cette figure représente la création d’une instance du modèle DAE et l’application de la méthode `applyModel()` pour le débruitage.

Application de la méthode `showSlice` (voir la figure 3.7)

```

for i in range(10):
    dae.showSlice(i)

```

FIGURE 3.7 – Méthode `showSlice()` de la classe DAE.

Résultat de la méthode showSlice() (voir la figure 3.8) :

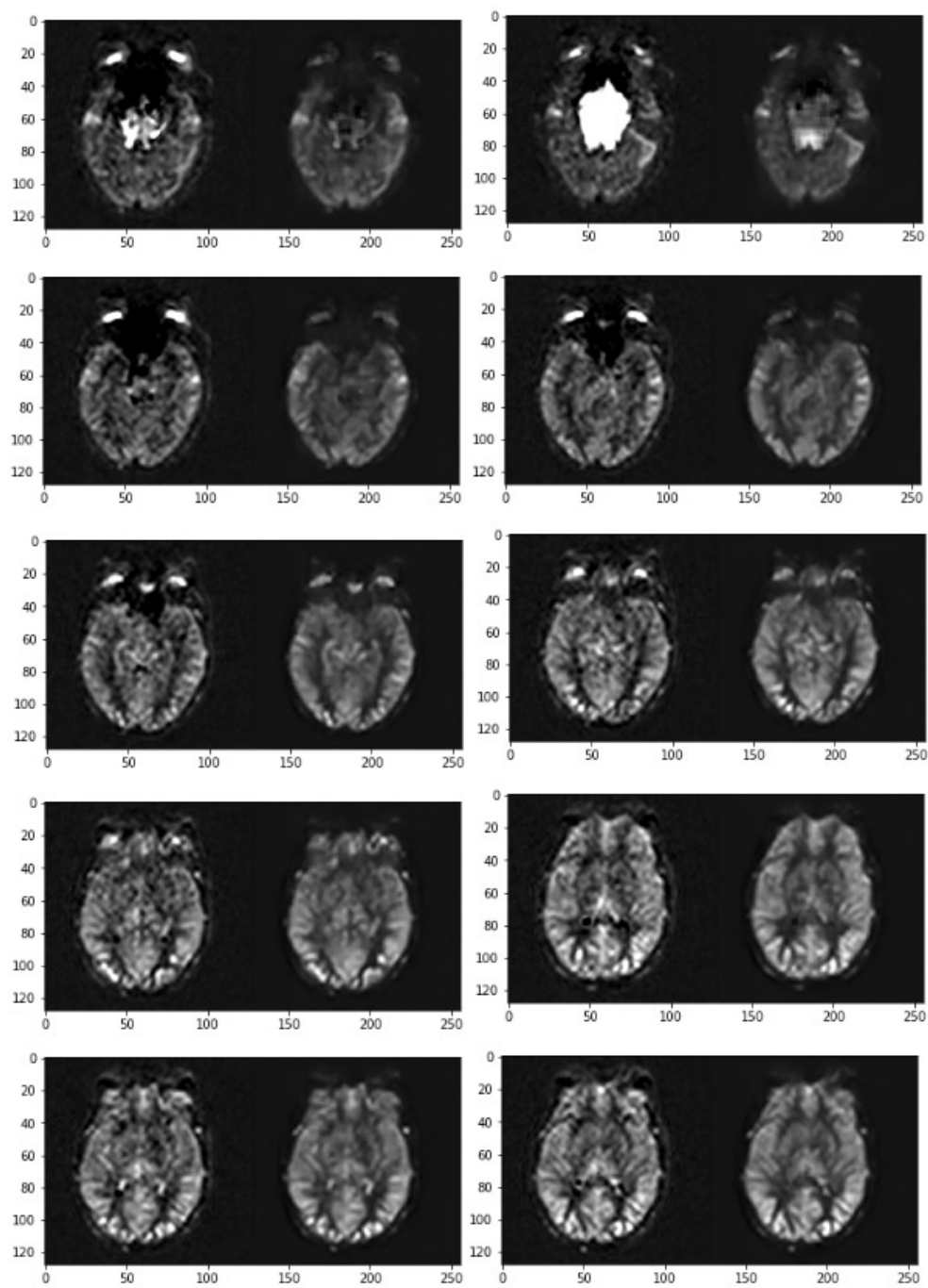


FIGURE 3.8 – Résultat de la méthode showSlice() pour le modèle DAE.

Application de la méthode `showTraining()` (voir la figure 3.9) :

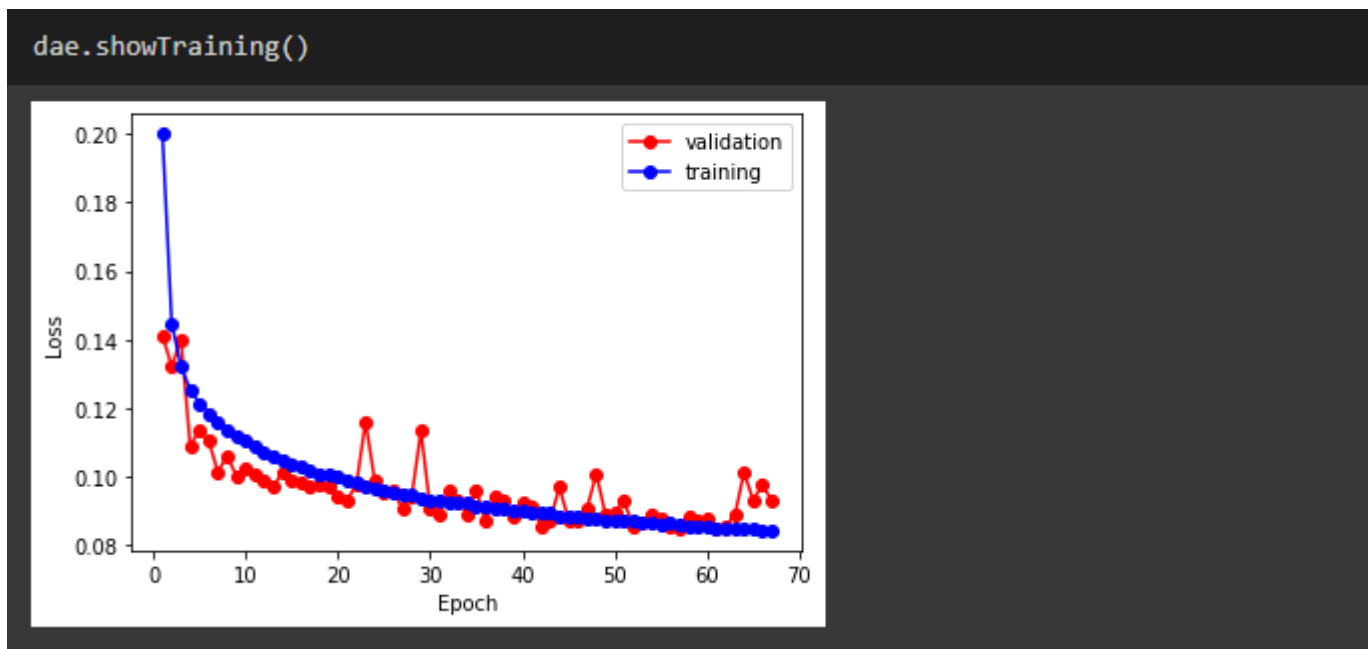


FIGURE 3.9 – Résultat de la méthode `showTraining()` pour le modèle DAE.

La figure au dessus représente l'application et le résultat de la méthode `showTraining()` qui donne un graphe de la perte de formation et de validation pendant l'entraînement du modèle.

Création du modèle DAE (voir la figure 3.10) :

```

Autoencoder=autoencoder()
Autoencoder.compile(optimizer='adam', loss='binary_crossentropy')
Autoencoder.summary()

Model: "model"

```

Layer (type)	Output Shape	Param #	Connected to
input_1 (InputLayer)	[(None, None, None, 1)]	0	[]
conv2d (Conv2D)	(None, None, None, 64)	640	['input_1[0][0]']
activation (Activation)	(None, None, None, 64)	0	['conv2d[0][0]']
max_pooling2d (MaxPooling2D)	(None, None, None, 64)	0	['activation[0][0]']
conv2d_1 (Conv2D)	(None, None, None, 64)	36928	['max_pooling2d[0][0]']
activation_1 (Activation)	(None, None, None, 64)	0	['conv2d_1[0][0]']
max_pooling2d_1 (MaxPooling2D)	(None, None, None, 64)	0	['activation_1[0][0]']
conv2d_2 (Conv2D)	(None, None, None, 64)	36928	['max_pooling2d_1[0][0]']
activation_2 (Activation)	(None, None, None, 64)	0	['conv2d_2[0][0]']
up_sampling2d (UpSampling2D)	(None, None, None, 64)	0	['activation_2[0][0]']
concatenate (Concatenate)	(None, None, None, 128)	0	['activation_1[0][0]', 'up_sampling2d[0][0]']
conv2d_3 (Conv2D)	(None, None, None, 64)	73792	['concatenate[0][0]']
activation_3 (Activation)	(None, None, None, 64)	0	['conv2d_3[0][0]']
up_sampling2d_1 (UpSampling2D)	(None, None, None, 64)	0	['activation_3[0][0]']
concatenate_1 (Concatenate)	(None, None, None, 128)	0	['activation[0][0]', 'up_sampling2d_1[0][0]']
conv2d_4 (Conv2D)	(None, None, None, 1)	1153	['concatenate_1[0][0]']

```

=====
Total params: 149,441
Trainable params: 149,441
Non-trainable params: 0

```

FIGURE 3.10 – Création du modèle DAE.

3.4 Création du modèle DnCNN

On a utilisé un dépôt en github «Beyond a Gaussian Denoiser : Residual Learning of Deep CNN for Image Denoising» [5] qui contient le code source nécessaire à la création du modèle DnCNN décrit.



3.4.1 Les modules utilisés

```
import os, glob, sys, time, datetime
import cv2
import argparse
import re
import numpy as np
import matplotlib.pyplot as plt
from keras.models import load_model, model_from_json
from skimage.metrics import peak_signal_noise_ratio
from skimage.metrics import structural_similarity
from skimage.io import imread, imsave
from keras.layers import Input, Conv2D, BatchNormalization, Activation, Subtract
from keras.models import Model, load_model
from keras.callbacks import CSVLogger, ModelCheckpoint, LearningRateScheduler
from keras.optimizers import adam_v2
import keras.backend as K
```

FIGURE 3.11 – Les modules utilisés pour le modèle DnCNN.

3.4.2 Description des principales fonctionnalités du code source

On a utilisé les méthodes suivantes :

show(x, title=None, cbar=False, figsize=None) permet de visualiser les images bruitées d'entrée en utilisant la librairie matplotlib.pyplot.

to_tensor(img) prend en paramètre une image d'entrée et la transformer à un tableau avec des valeurs de type float.

from_tensor(img) prend en charge une image pour supprimer les entrées unidimensionnelles de la forme d'un tableau.

DnCNN(depth, filters=64, image_channels=1, use_bnorm=True) permet de retourner le modèle appliqué aux données avec ses couches de convolution.

3.4.3 Application du code source

Tout d'abord, on a importé le modèle entraîné pour le débruitage gaussien décrit dans [5] (voir la figure 3.12) :

```
from google.colab import files

uploaded = files.upload()

for fn in uploaded.keys():
    print('User uploaded file "{name}" with length {length} bytes'.format(
        name=fn, length=len(uploaded[fn])))

Saving log.csv to log.csv
Saving model.h5 to model.h5
Saving model.json to model.json
User uploaded file "log.csv" with length 0 bytes
User uploaded file "model.h5" with length 2349584 bytes
User uploaded file "model.json" with length 21926 bytes
```

FIGURE 3.12 – Code source de l’importation du modèle entraîné pour le débruitage gaussien.

On a appliqué le modèle DnCNN sur les dix images d’entrée (voir la figure 3.13) :

```
x = np.array(imread(os.path.join(args.set_dir)), dtype=np.float32) / 255.0
y = x + np.random.normal(0, args.sigma/255.0, x.shape) # Add Gaussian noise without clipping
y = y.astype(np.float32)
y_ = to_tensor(y)
x_ = model.predict(y_) # inference
x_=from_tensor(x_)
```

FIGURE 3.13 – Application du modèle DnCNN.

On affiche les images de sortie par la méthode show() (voir la figure 3.14) :

```
show(np.hstack((y,x_))) # show the image
```

FIGURE 3.14 – Méthode show() de DnCNN.

Résultat de la méthode show() (voir la figure 3.15) :

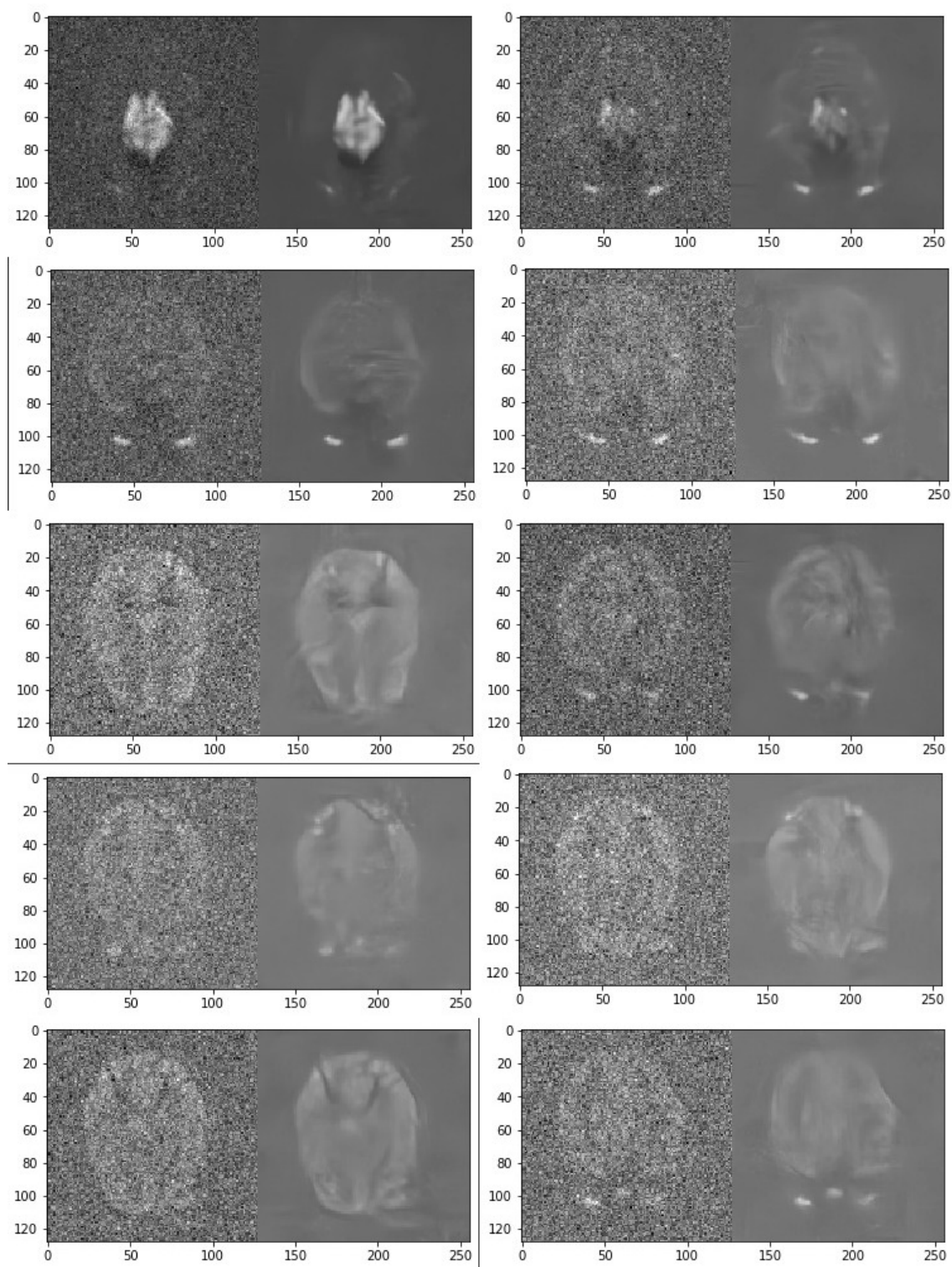


FIGURE 3.15 – Application de la méthode `show()` de DnCNN.

3.5 Création du modèle filtrage non local moyen

3.5.1 L'environnement de développement utilisé

On va appliquer une implémentation en ligne de l'algorithme de débruitage NLmeans. Ce code source est écrit en langage C et C++, il lit et écrit des images PNG, mais peut être facilement adapté à tout autre format de fichier.



3.5.2 Description des principales fonctionnalités du code source

On a utilisé la version en ligne de l'algorithme Non Local Means Denoising [6] en suivant les étapes suivantes :

Tout d'abord, on a importé les dix images de la séquence ASL utilisée une par une (voir la figure 3.16).

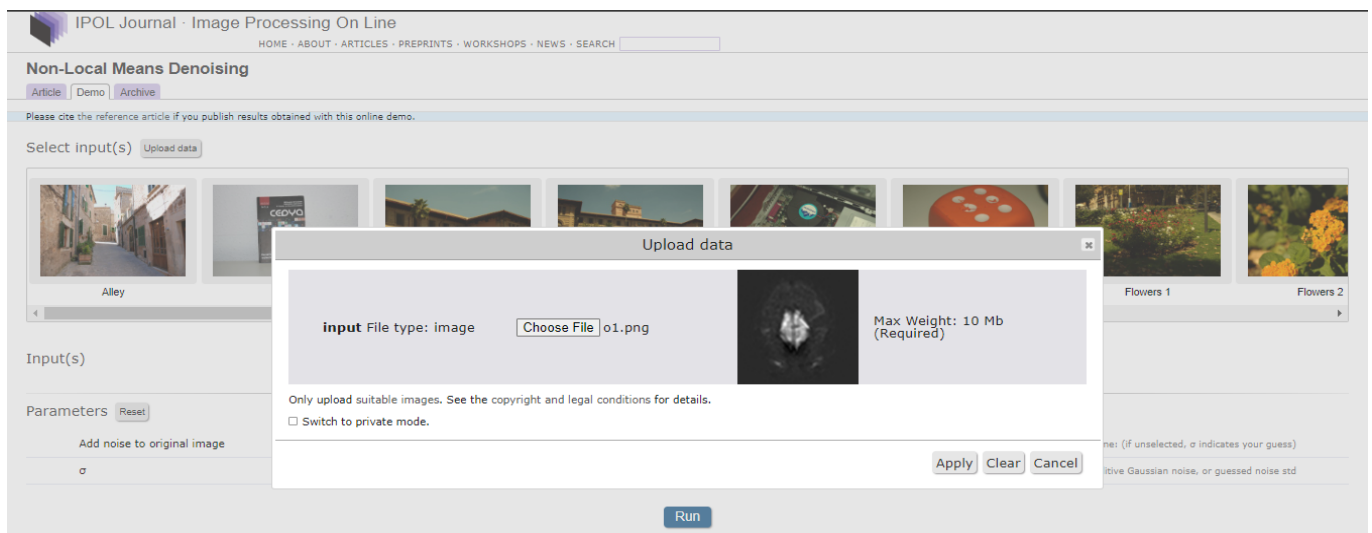


FIGURE 3.16 – Importation des images bruitées dans l'implémentation en ligne du NLmeans.

Après, on a ajouté un bruit minimal σ , dans notre cas $\sigma=1$ (voir la figure 3.17)

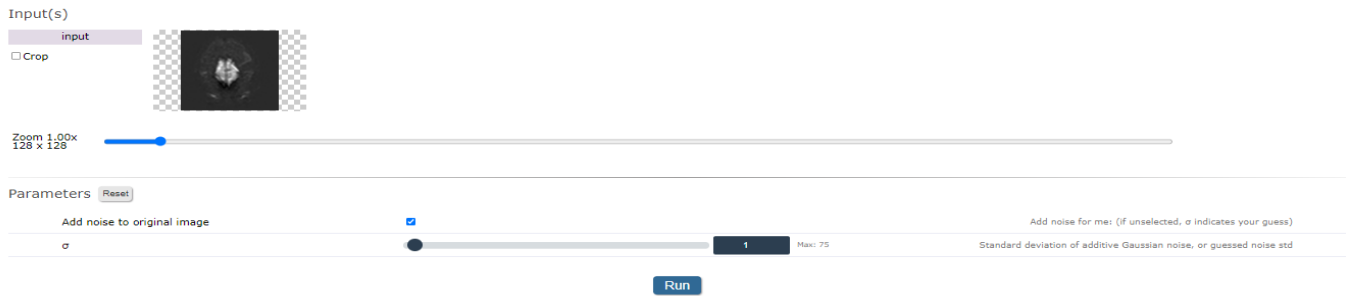


FIGURE 3.17 – Initialisation de σ dans l'implémentation du NLmeans.

Enfin, l'algorithme retourne les images débruitées une par une (voir la figure 3.18).

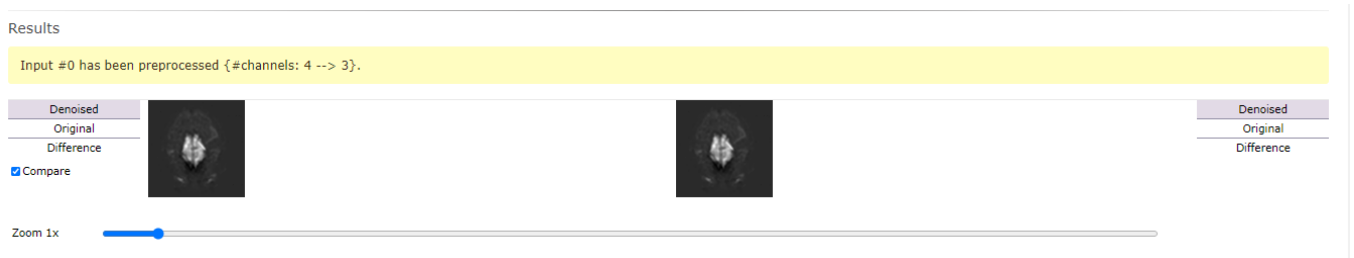


FIGURE 3.18 – Application du NLmeans.

Dans notre cas, on a obtenu les dix images débruitées suivantes (voir la figure 3.19) :

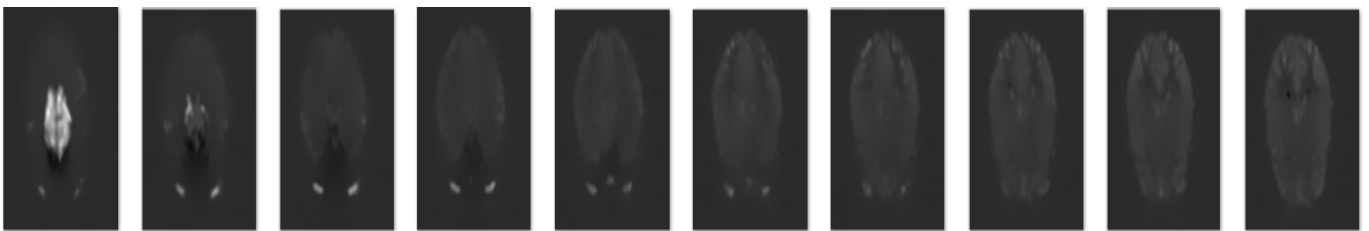


FIGURE 3.19 – Images débruitées dans l'application du NLmeans.

3.6 Visualisation des images débruitées par les trois algorithmes

Dans cette partie, on visualise les images débruitées par les trois algorithmes (voir la figure 3.20 et la figure 3.21).

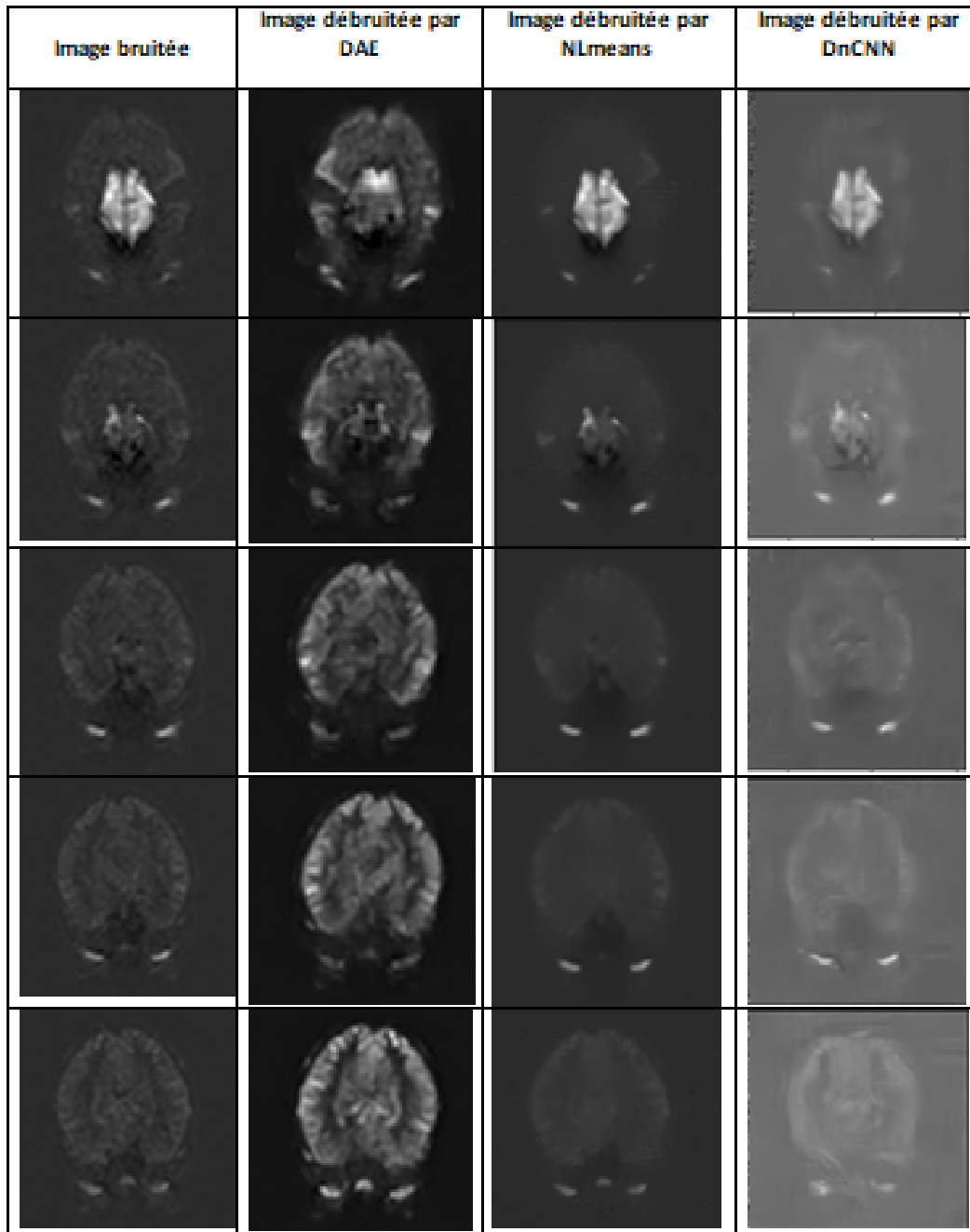


FIGURE 3.20 – Les images débruitées par les trois algorithmes.

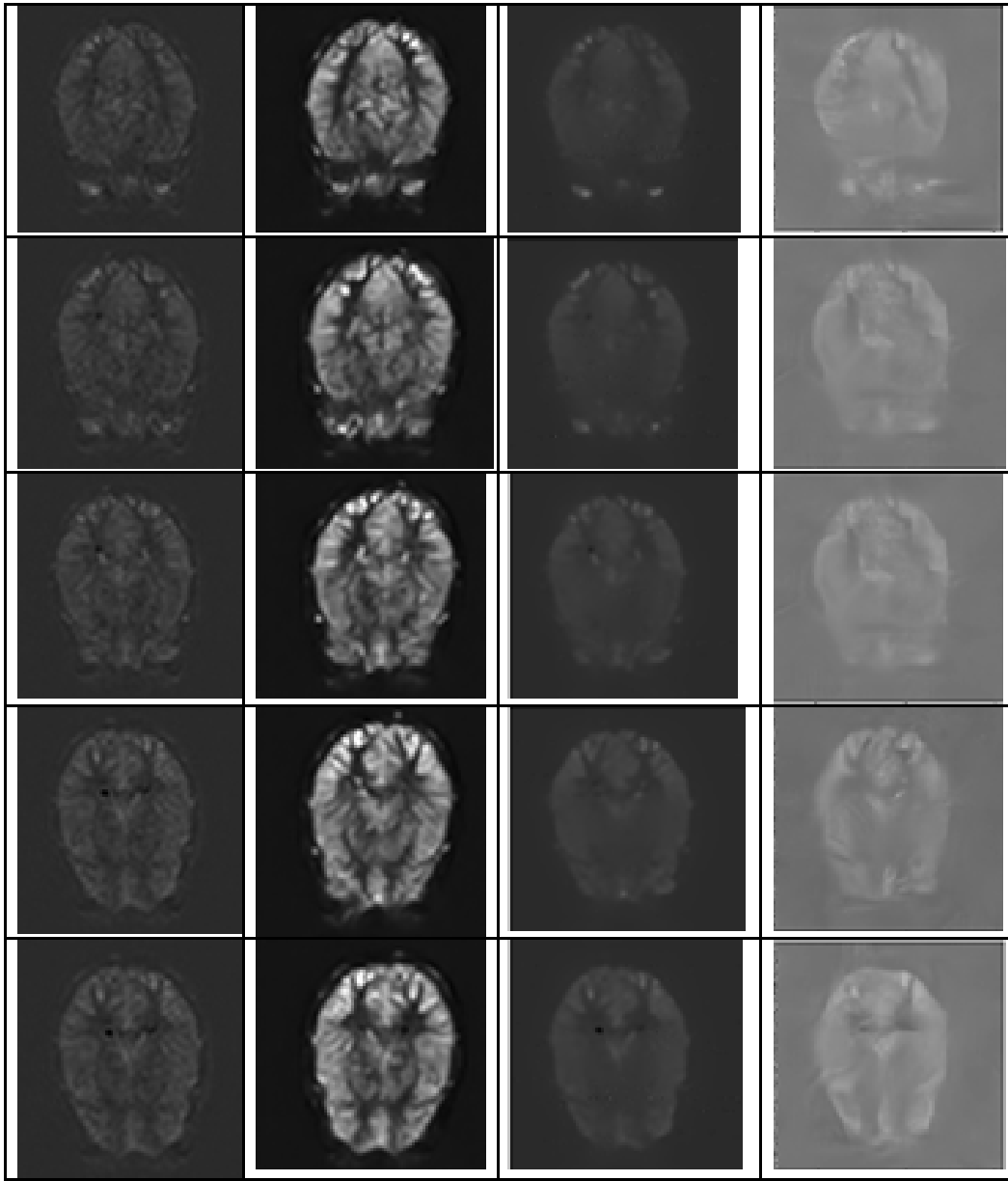


FIGURE 3.21 – Suite d’images débruitées par les trois algorithmes.

3.7 PSNR «Peak Signal to Noise Ratio»

3.7.1 Définition

Le PSNR est une mesure prise en image numérique, tout particulièrement en compression d'image. Cette mesure permet d'évaluer la qualité d'une image compressée que l'origine. Le PSNR est exprimé en termes d'échelle logarithmique de décibels, il est utilisé comme mesure de la qualité pour le type de compression lossy comme pour la JPG. En effet, la valeur du PSNR d'une image dépend de la qualité de l'image :

La formule mathématique du $PSNR = 20 * \log_{10}(\frac{\max_f}{\sqrt{MSE}})$

3.7.2 Mesure de qualité en PSNR du modèle DAE

On a utilisé la méthode PSNR qui retourne le PSNR des données d'entrées et données de sorties en utilisant la formule mathématique du PSNR (voir la figure 3.22).

```
import math
import os
import numpy as np

def psnr(original, contrast):
    mse = np.mean((original - contrast) ** 2)
    if mse == 0:
        return 100
    PIXEL_MAX = 255.0
    PSNR = 20 * math.log10(PIXEL_MAX / math.sqrt(mse))
    return PSNR

def main():
    for i,j in zip(range(11), range(11)):
        original=dae.dMraw[i]
        contrast=dae.dMdenoised[j]
        print(f"PSNR value is {psnr(original, contrast)} dB")

if __name__ == '__main__':
    main()
```

FIGURE 3.22 – Code source de la fonction PSNR.

On a obtenu les résultats cités dans la figure 3.24.

3.7.3 Mesure de qualité en PSNR du modèle DnCNN

Le code source utilisé inclut la méthode qui calcule le PSNR (voir la figure 3.23) :

```
psnr_x=peak_signal_noise_ratio(x,x)  
print(psnr_x_)
```

FIGURE 3.23 – Calcul du PSNR dans le DnCNN.

3.7.4 Mesure de qualité en PSNR du modèle NLmeans

Tel que mentionné précédemment, la version en ligne de cet algorithme a été utilisée. En effet, les dix images de la séquence ASL ont été débruitées et le PSNR de chaque image d'entrée a été obtenu par rapport à son image de contraste. (voir la figure 3.24).

3.7.5 Comparaison de la performance des trois algorithmes DAE, NLmeans et DnCNN

En mesurant les PSNR des dix images bruitées figurées dans 3.2 par rapport aux images débruitées figurées respectivement dans 3.20 et 3.21 par les trois algorithmes de débruitage utilisés, on a obtenu les résultats suivants (voir la figure 3.24) :

PSNR en (dB) avec DAE	PSNR en (dB) avec NLmeans	PSNR en (dB) avec DnCNN
45.95	38.64	35.53
48.25	37.83	35.89
48.43	37.94	36.29
47.35	37.78	36.23
47.85	37.24	35.72
47.71	37.25	35.76
47.44	37.33	35.64
47.04	37.09	35.50
46.79	37.44	35.42
46.69	37.30	35.50

FIGURE 3.24 – Comparaison des PSNR retournés de chaque algorithme.

En se basant sur le tableau de la figure 3.25, on remarque que les PSNR obtenus par l'algorithme de DAE varient entre 45 et 49 ce qui signifie que la qualité de ces images est très bonne, par contre on remarque les PSNR obtenus par le filtrage non local varient entre 37 et 39 ce qui signifie que la qualité des images est bonne, ainsi les PSNR obtenus par le DnCNN varient entre 35 et 37 ce qui signifie que la qualité des images est bonne, ceci permet de conclure que le modèle DAE est le meilleur algorithme pour débruiter et améliorer la qualité des images ASL bruitées.

<i>Qualité</i>	<i>Très bonne</i>		<i>Bonne</i>		<i>Acceptable</i>		<i>Mauvaise</i>
<i>PSNR (dB)</i>	<i>>40</i>	<i>≈40</i>	<i>35-40</i>	<i>≈35</i>	<i>30-35</i>	<i>≈30</i>	<i><30</i>

FIGURE 3.25 – PSNR en fonction de qualité d'image.

3.8 Conclusion

D'après l'analyse de performance des trois algorithmes, on conclut que le modèle DAE est plus efficace pour le débruitage des images acquises par marquage des spins artériels.

Conclusion générale

Notre étude avait pour ambition de réduire le bruit des images acquises par marquage des spins artériels en IRM. Tout au long de la préparation de notre projet de fin d'études, on a essayé de mettre en pratique les connaissances acquises durant nos études universitaires et cela dans le but de déterminer si les techniques de débruitage de l'intelligence artificielle surpassent les techniques standard en termes de réduction du bruit.

Au cours de cette étude, on a étudié la technique du marquage des spins artériels et ses différents types, on a testé des algorithmes de débruitage comme un algorithme d'autoencodeur débruiteur DAE, un algorithme d'apprentissage du réseau de neurones convolutifs de débruitage DnCNN et un algorithme classique du filtrage non local moyen NLmeans.

Le critère de comparaison pris en considération est la qualité des images débruitées par les trois algorithmes qui est mesurée à l'aide de l'indicateur d'évaluation de qualité de l'image PSNR. D'après l'étude comparative qu'on a réalisée, on a remarqué que la qualité des images débruitées par DAE est meilleure par rapport à la qualité des images débruitées par DnCNN et l'algorithme NLmeans.

Comme perspective, on souhaite pour les prochains projets de fin d'études d'élargir notre travail sur la réalisation d'une application web qui utilise cet algorithme DAE pour le débruitage des images acquises par marquage des spins artériels.

Bibliographie

- [1] Marjorie Villien, "*Méthodologie et application de l'imagerie de la perfusion cérébrale et de la vasoréactivité par IRM*", Université de Grenoble, 229 :38-42. (2012).
- [2] Clément Debacker, "*Développement de l'imagerie de perfusion cérébrale par marquage des spins artériels*", Université de Grenoble, 149 :30-45. (2014).
- [3] Patrick W. Hales, Josef Pfeuffer and Chris A. Clark, "*Combined Denoising and Suppression of Transient Artifacts in Arterial Spin Labeling MRI Using Deep Learning*", J. MAGN. RESON. IMAGING, 14 :1-4. (2020).
- [4] Patrick Hales, (2020) "*Combined Denoising and Suppression of Transient Artifacts in Arterial Spin Labeling MRI Using Deep Learning*". Disponible sur : <https://github.com/patrickhales/asl-denoising> (Consulté le : 16 janvier 2020).
- [5] cszn , (2021) *Beyond a Gaussian Denoiser : Residual Learning of Deep CNN for Image Denoising*. Disponible sur : <https://github.com/cszn/DnCNN> (Consulté le : 09 octobre 2021).
- [6] Antoni Buades, Bartomeu Coll, Jean-Michel Morel.(2011) *Non-Local Means Denoising*. Disponible sur : https://doi.org/10.5201/ipol.2011.bcm_nlm (Consulté le : 13 septembre 2011).
- [7] *ImageJ*. Disponible sur : <https://imagej.nih.gov/ij/download.html>.