REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE

MINISTERE DE L'ENSEIGNEMENT SUPERIEUR ET DE LA RECHERCHE SCIENTIFIQUE

Université de Mohamed El-Bachir El-Ibrahimi - Bordj Bou Arreridj

Faculté *des Sciences et de la Technologie*

Département d'Electronique

Mémoire

Présenté pour obtenir

LE DIPLOME DE MASTER

FILIERE : **Eléctronique.**

Spécialité : Industries électroniques.

Par

- **TERAI Salim.**

Intitulé

Algorithme Hybride de Débruitage Image/Vidéo à base de Réseaux de Neurones Convolutionnels (CNN).

Evalué le : 14-09-2021

Par la commission d'évaluation composée de* :

Nom & Prénom	*Qualité*	*Etablissement*
Dr. BOUDECHICHE Djamel	*Président*	*Univ-BBA*
Dr. SID AHMED Soumia	*Encadreur*	*Univ-BBA*
Pr. MESSALI Zoubeida	*Co-encadreur*	*Univ-BBA*
Dr. HACINE GHARBI Abdenour	*Examinateur*	*Univ-BBA*

Année Universitaire 2020/2021

---

* Conformément à :

- L'arrêté n°055 du 21 janvier 2021 Fixant dispositions exceptionnelles autorisées en matière d'organisation et gestion pédagogique, de l'évaluation et de la progression des étudiants, durant la période COVID-19 au titre de l'année universitaire 2020-2021 ;
- Procès-verbal de la réunion de l'équipe du domaine des Sciences et Technologies du mois de Mai 2021.

Remerciements

Avant toute chose, je remercie **«Allah »**, qui nous aide et nous donne la volonté, le courage et la patience durant ces années d'étude.

Je souhaite adresser mes remerciements les plus chères aux personnes qui ont contribués à l'élaboration et l'avancement de mon sujet d'étude, ainsi que l'ensemble de la famille universitaire de Bordj-Bou-Arreridj et plus précisément ceux du département d'Electronique.

Je remercie mes encadrantes **Dr.SID AHMED Soumia**, et **Prof MESSALI Zoubeida** qui ont toujours montré leurs intérêts et leur suivi de ce travail et qui ont été disponibles tout au long de mon travail, sans oublier leurs remarques pertinentes et leurs conseils et patiences durant l'étude de ce sujet pour aboutir à un travail de qualité.

Je désir aussi remercier les enseignants de l'université Mohamed Elbachir El Ibrahimi, qui nous ont fourni les outils nécessaires à la réussite de nos études universitaires pendant mon cursus.

Enfin je tiens à exprimer toute ma gratitude à l'ensemble de ma famille et proches pour leur soutien tout au long de ces années.

Table des matières

Chapitre 3 : Implémentation des Algorithmes et Résultats des Expériences

Liste des figures

Liste des Tableaux

Liste des Acronymes

AI	Artificial Intelligence.
BM3D	Block Matching 3D.
CNN	Convolutional Neural Network.
DnCNN	Discriminative Convolutional Neural Network
DVDnet	Deep Video Denoising network.
FastDVDnet	Fast Deep Video Denoising network.
MLP	Multi Layer perceptron.
PSNR	Peak Signal Noise Ratio.
RMSE	Root mean square error.
SLP	Single Layer Perceptron.
SSIM	Structural Similarity Index.
VBM4D	Video Block Matching 4D.
VNLB	Video Non-Local Bayes.

Introduction Générale

Introduction Générale

Dans notre monde moderne, le monde du numérique où la technologie se développe à grande échelle, nous avons l'intelligence artificielle qui a fait son grand pas en se montrant comme un atout majeur au sein de la société moderne.

L'intelligence artificielle est un vaste domaine technologique qui accompagne des branches diverses et variées. Fondée en 1956 par John McCarthy avec son équipe de recherche dont un parmi eux très connu, Alan Turing qui par la suite, fera émergé les principes de l'intelligence artificielle [1].

Dans ce manuscrit nous nous sommes intéressés à l'apprentissage profond (Deep Learning) qui peut s'appliquer dans notre domaine technique et sur les systèmes embarqués notamment, le traitement d'images ou aussi appelé imagerie dans les multimédias. La tâche que nous voudrions apporter à partir d'un apprentissage profond est l'utilisation d'un réseau profond le réseau de neurones convolutionel CNN (Convolutional Neural Network) sur des images et vidéos bruitées. L'objectif dans ce travail de master est d'exploiter les performances du réseau profond CNN pour enlever le bruit qui affecte les images/vidéo. Comparé au débruitage d'image, le débruitage vidéo reste un domaine assez récent et peu exploré.

Par le biais de nouvelles techniques et méthodes de débruitage d'images et vidéo basées sur des techniques d'apprentissage profond, plusieurs algorithmes seront présentés et implémentés dans ce manuscrit afin de faire une étude comparative, quantitative et qualitative en termes de calcul des critères usuels, à savoir le PSNR, SSIM, RMSE. L'originalité de notre travail réside dans l'implémentation d'algorithmes de débruitage d'images/vidéo à base de CNN.

Le mémoire est organisé comme suit :

- Dans le Chapitre 1, nous allons introduire les principes de l'apprentissage profond, et les vocabulaires et notions de bases de celui-ci.
- Dans le Chapitre 2, nous présentons les algorithmes de débruitage pour le cas des images et vidéos.
- Dans le Chapitre 3, nous détaillons l'implémentation des algorithmes considérés d'images et vidéos ainsi que les architectures des techniques et algorithmes utilisés et

tous les tests que nous avons effectués avec des discussions des résultats. Plus précisément nous allons implémenter 03 algorithmes de débruitage images qui sont **BM3D**, **DnCNN**, **FFDnet**, et 05 algorithmes de débruitage vidéo qui sont **SPTWO**, **VNLB**, **VBM4D**, **DVDnet**, **FastDVDnet**. Les algorithmes conventionnels de débruitage d'images se basent essentiellement sur le débruitage par **Patches**, et les algorithmes de débruitage vidéo se basent sur **l'estimation du mouvement**, ajouter aussi les CNN pour certains. Les algorithmes sont appliqués sur plusieurs jeux de données pour montrer les performances de ces algorithmes. Nous considérons le cas du bruit blanc gaussien additif. Nous utilisons plusieurs environnements dans notre étude de simulation. La préparation du jeu de données est une étape cruciale. Pour cela, nous utilisons des logiciels bien specifiques au traitement des vidéos, à savoir **FFMPEG.**

Une conclusion générale ainsi que les perspectives sont présentés à la fin du manuscrit pour poursuivre les recherches effectuées.

Chapitre 1

Réseaux de Neurones Profonds

Résumé

Dans ce chapitre, nous allons détailler l'apprentissage profond et ses constituants et nous familiariser avec les différents vocabulaires qui seront les outils fondamentaux pour la création d'un réseau de neurones profond en Deep Learning.

Sommaire

1.1. Introduction

Nous avons mentionné dans l'introduction générale que l'intelligence artificielle comportait diverses branches, celles-ci consistent en plusieurs aspects d'autonomie de la machine ou plus exactement, un phénomène dont l'apprentissage devient autonome.

Parmi ces branches, nous avons celle du Machine Learning qui veut dire apprentissage automatique de la machine. Ce même domaine peut lui aussi se diviser en tant qu'apprentissage supervisé, non supervisé, ou aussi apprentissage par renforcement. En vue des développements qui ont suivi cette discipline, les recherches ont rapporté un autre aspect de l'apprentissage qui a vu le jour dans les années après (presque 30 ans après l'apparition du Machine Learning) appelé, apprentissage profond ou Deep Learning [2]. La figure1.1 illustre la position du Deep Learning au sein du domaine de l'intelligence artificielle aussi dénommé AI (pour Artificial Intelligence).

Nous allons donc dans ce chapitre détailler les principes de base d'un réseau profond et ses principales caractéristiques qui font son importance dans le domaine du traitement d'image.

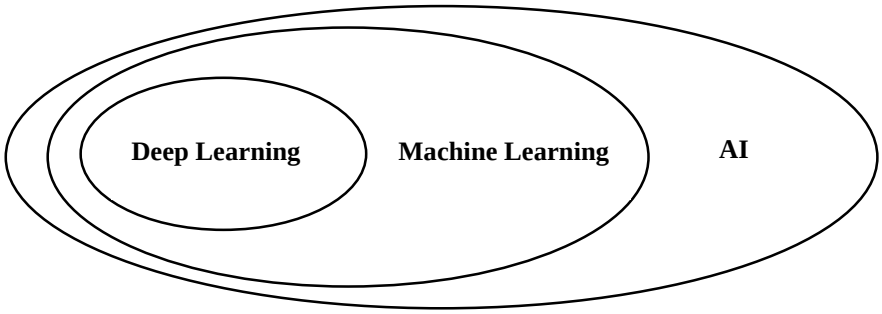


Figure 1.1: Hiérarchie et position du Deep Learning par rapport au Machine Learning et IA.

1.2. Réseau de Neurones Artificiels

Avant de comprendre ce qu'est un réseau de neurones (Neural Network) artificiel, il faut tout d'abord comprendre ce qu'est le Maching Learning, grâce à quelques généralités toutes simples sans détailler étant donné que c'est le Deep Learning qui nous intéresse dans notre travail, nous pourrons ainsi comprendre la suite qui concerne le réseau de neurones ainsi que le Deep Learning.

Le Machine Learning peut être caractérisé par un modèle mathématique qui suit des données d'apprentissage, on obtient ce dernier par des entrées de données X et des sorties cibles Y pour le cas d'un apprentissage supervisé (ce qui n'est pas toujours le cas). À partir de là, la machine obtient plus d'expérience en se familiarisant avec ces données appelées « **Dataset »** [3], à partir de ces données Dataset(X , Y) où X représente les caractéristiques nommées « **Features »** et Y la cible « **Target** », leur importance réside en tant qu'information utile qui sera utilisée par le modèle via le concept de « **Training »** qui lui permettra donc par le biais de cet entrainement de tracer son modèle qui va pouvoir représenter le processus d'apprentissage automatique [2], [3]. Plus les features concernés sont bien définis plus le modèle sera précis [3]. La figure suivante montre un exemple de Machine Learning qui va avoir en résultat un modèle :

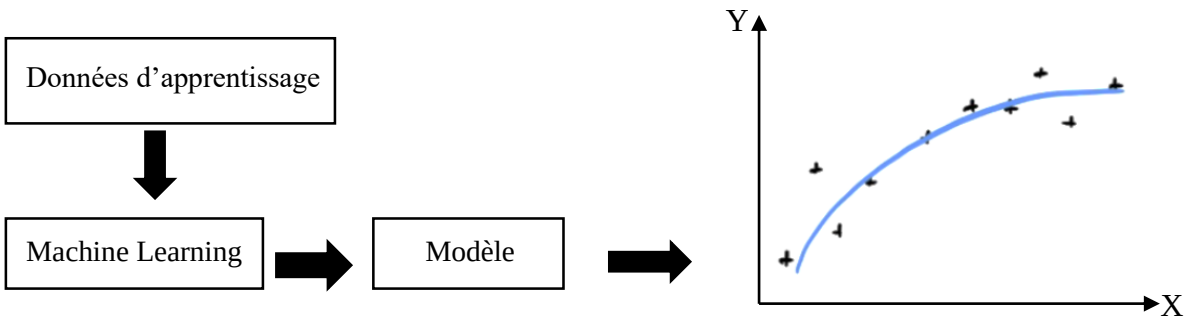


Figure 1.2: Ce qui se passe pendant le processus d'apprentissage.

Donc comme cité, les données d'apprentissage vont influencer sur la valeur de y par le facteur feature qui va résulter d'un modèle comme illustré sur la figure 1.2 en bleu où l'espace qu'il y'a entre le modèle en bleu avec les points dans l'espace représente la précision (Accuracy) ou l'erreur entre notre modèle et les données. Ceci constitue donc une fonction de prédiction $f(X)$ pour minimiser les erreurs via un algorithme d'optimisation [4].

Ce type de calcul pour la machine est très efficace pour prédire des calculs automatiques pour les statistiques ou la science de données en général, mais supposant que nous avons une large quantité de données comme des centaines de milliers d'images à faire entrer avec de grosses tailles, on aura donc une très large quantité de pixels, ce qui rend le calcul trop grand voir même limité en machine Learning. C'est pourquoi vient alors la notion de réseau profond afin de traiter ce genre de données et de l'exploiter dans le Deep Learning [5].

Nous pouvons ainsi traduire cette notion dans la figure suivante comme suit :

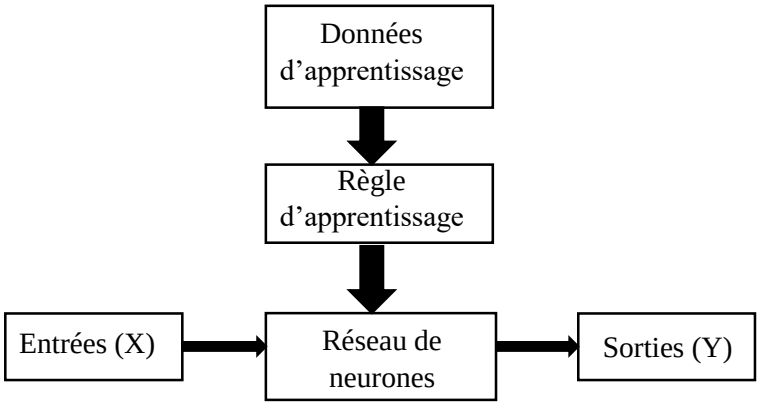


Figure 1.3: Relation entre Machine Learning et Deep Learning.

De de là, notre modèle en Machine Learning est remplacé par un réseau de neurones et la règle d'apprentissage prend place au Machine Learning. Dans ce contexte de réseau de neurones, la règle d'apprentissage fera office de processus à déterminer le modèle (réseau de neurones) [2]. Il faut aussi faire la différence entre les données d'apprentissage qui sont les données sur qui le modèle s'entraîne, et les données d'entrées qui seront les données à prédire en sortie.

1.2.1. Perceptron

Le perceptron est en quelque sorte le niveau le plus bas et le plus simple à réaliser qu'un réseau de neurones artificiel puisse avoir. Ceci vient du fait qu'un neurone artificiel est tiré d'un neurone naturel en faisant le rapprochement comme le montre la figure 1.4 suivante :

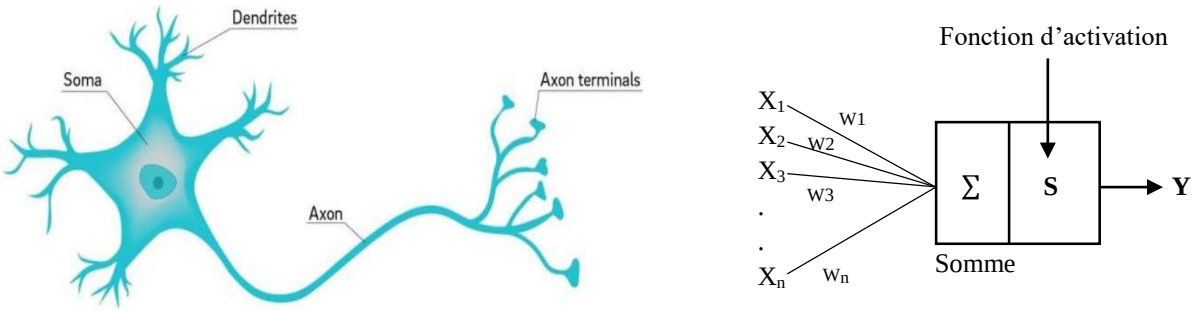


Figure 1.4: a. Neurone biologique [3] b. Neurone artificiel

Un neurone biologique fait passer des informations au travers des **synapses**[6] dans **la dendrite**, vers d'autres neurones biologiques, le même principe est adopté pour un neurone

artificiel qui fera office d'entrés de données X vers la sortie Y (**Axon terminals**); cette transmission d'information est provoquée par une excitation ou inhibition grâce aux stimulus qu'il reçoit, en faisant l'analogie avec un neurone artificiel, cela se traduit par **une fonction de transfert**, aussi appelée **fonction d'activation** [3]. La transformation appliquée par la fonction d'activation de la donnée se fait par seuillage (**Thresholding**) afin de déterminer une fonction de décision [7].

En regardant bien dans la figure 1.4 sur le neurone artificiel on a une combinaison linéaire qui s'opère tel que :

$$S = X_1 \cdot W_1 + X_2 \cdot W_2 + X_3 \cdot W_3 + B \tag{1.1}$$

$$Y = F(S) \tag{1.2}$$

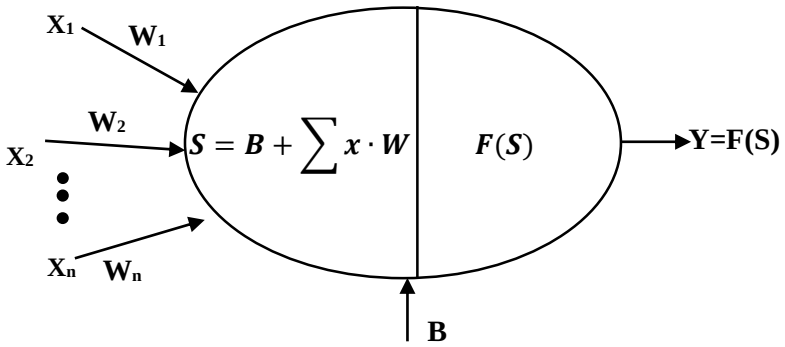


Figure 1.5: Relation de linéarité dans un Perceptron.

De là commence le principe d'apprentissage de base grâce aux poids synaptiques $\boldsymbol{Wn}$ et le biais $\boldsymbol{B}$ entrant en phase d'agrégation (Equation (1.1)) et en phase d'activation (Threshold). Chacun de ces deux influe sur le comportement du perceptron et donc le modèle en sortie [8], soit par un offset par le biais ou changement de magnitude voir même le signe par les poids synaptiques, étant donné que nous avons une sorte de linéarité qui ressemble à une droite $\boldsymbol{F(X) = aX + B}$. Cette influence sur le comportement du modèle pour arriver à la sortie désirée se fait par une mise à jour des paramètres (Weights update) [2], c'est pourquoi les paramètres Wn et B sont des paramètres qui entrent dans l'entraînement de la machine ou plus précisément l'algorithme d'apprentissage qui suit l'équation suivante :

$$W = W + \alpha\left(Y_{ref} - Y\right)X \tag{1.3}$$

À travers ce genre de mise à jour que se fait une sorte de réglage automatique via cet algorithme qui permet de renforcer les paramètres des poids synaptiques à chaque fois qu'une entrée X est activée en même temps que la sortie Y présente dans ces données [2][3].

Ceci dit, cela reste très limité en termes de ce que doit apporter le domaine du réseau de neurones, c'est pourquoi les fonctions d'activation aident à avoir un modèle qui entre dans la non-linéarité, nous avons par exemple des fonctions très connues comme la fonction sigmoïde par exemple définie tel que :

$$F(S) = \frac{1}{1+e^{-S}} \tag{1.4}$$

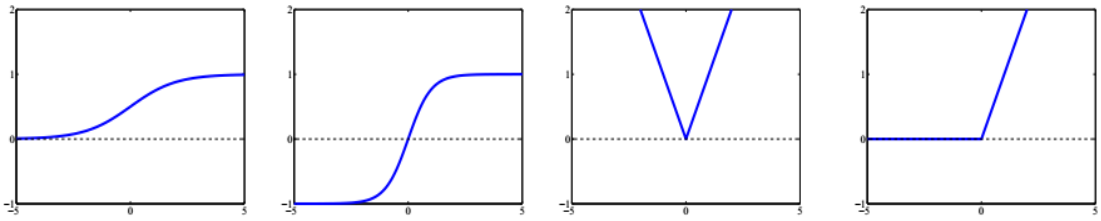


Figure 1.6: Exemple de fonctions non linéaires appliqués dans un perceptron.

Selon la figure 1.6 nous pouvons citer les fonctions en partant de gauche, la fonction logistique (équation 1.4) (Sigmoïde), la fonction tangente hyperbolique (équation 1.5), la fonction valeur absolue (équation 1.6), et la fonction Unité Linéaire Rectifiée (**ReLU** pour **Rectified Linear Unit**) qu'on aura l'occasion de voir dans le prochain chapitre qui sera utilisé dans notre travail (équation 1.7).

$$Tanh(S) = \frac{1-e^{-2(S)}}{1+e^{-2(S)}} \tag{1.5}$$

$$abs(S) = |S| \tag{1.6}$$

$$F = max(0, S) \tag{1.7}$$

Avec bien sûr, **S** qui représente les (weighted sum) c'est-à-dire la somme des produits entre la matrice des poids synaptiques **W** et les entrées du réseau en ajoutant aussi le vecteur biais **B**. On utilise les matrices pour simplifier les calculs mathématiques.

1.2.2. Perceptron Multi Couche (Multilayer Perceptron)

Cette contrainte de modèle linéaire du perceptron comme le problème logique XOR sur l'impossibilité séparation linéaire [9], qui le rend peu utile pour des phénomènes de vie courante

qui sont non-linéaires, il peut être résolu par le développement du Perceptron multi couche, en connectant plusieurs neurones ensemble, il est possible de résoudre des problèmes plus complexes.

Nous pouvons pour le cas d'un perceptron l'adapter avec plusieurs perceptrons en une seule couche (Figure 1.7), par exemple, pour classifier des images de multiples classes différentes en sorties. Pour cela en connectant les entrées avec toutes les sorties et que chaque perceptron de cette couche de sortie ait son propre biais ***B*** [10]. La figure suivante montre la différence entre un réseau de perceptrons mono couche, et un réseau de neurones multi couches

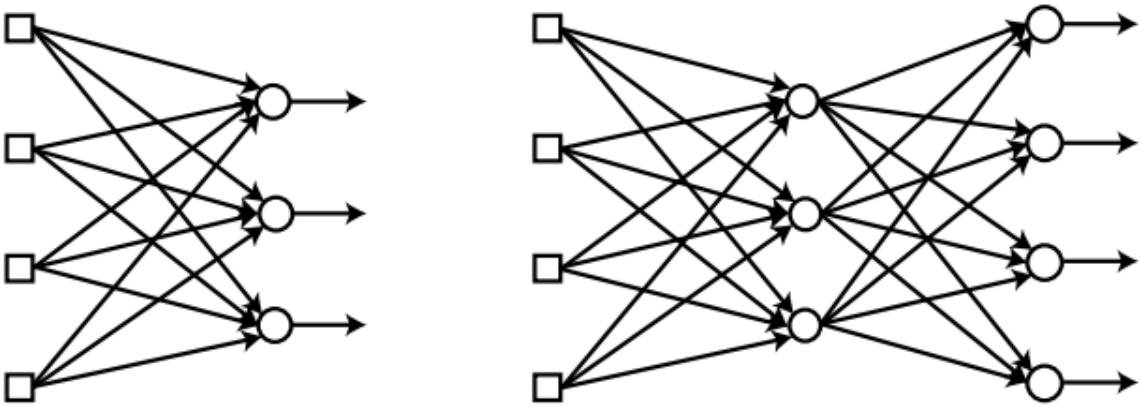


Figure 1.7: Différence entre réseau de neurones à une seule couche et multi couches [2].

Dans le cas d'un **SLP** (**S**ingle **L**ayer **P**erceptron à gauche de la figure) on peut avoir un seul perceptron comme on peut avoir plusieurs perceptrons dans une seule et unique couche [10], c'est pourquoi on fait la différence entre SLP et **MLP** (**M**ulti **L**ayer **P**erceptron à droite de la figure).

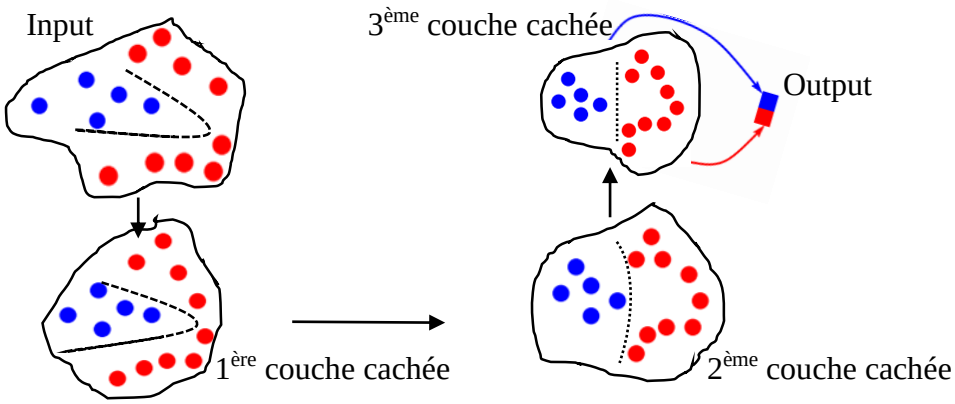


Figure 1.8: Projection des données dans un réseau multi couches.

Afin de mieux comprendre le réseau de perceptrons Multi-couches, la figure 1.8 illustre ce qui se passe lorsque les données traversent plusieurs couches qui sont traîtées de manière non linéaire en fonction des caractéristiques extraites des données par l'algorithme d'apprentissage au fur et à mesure que celles-ci se rapproche d'une couche à une autre vers la sortie qui devient linéaire[10]. À partir de cette idée, nous pouvons maintenant comprendre ce qu'est la profondeur d'un réseau de neurones.

En insérant des couches intermédiaires entre la couche d'entrée et de sortie (Figure 1.9) nommées Hidden layers en français couches cachées ; nous avons chaque neurone reçoit à son entrée les sorties des neurones de la couche précédente ; en d'autres termes, la couche de sortie reçoit en entrée les sorties de la dernière couche cachée, et celle-ci reçoit à son tour les sorties de la couche d'entrée précédente; c'est ce qu'on appelle réseau de propagation avant (**Feed Forward network)**, c'est à dire que nous n'avons pas de retour d'une couche vers la couche qui l'a précède [7], l'information des données démarre depuis la couche d'entrée vers la couche de sortie par l'intermédiaire des couches Hidden layers. Lorsque le nombre de couches cachées augmente alors ici on parle de Deep Learning car c'est à travers cette profondeur que l'exploitation des caractéristiques des données devient intéressante, c'est pourquoi les branches d'un réseau de neurones dépendent de l'architecture des couches [11].

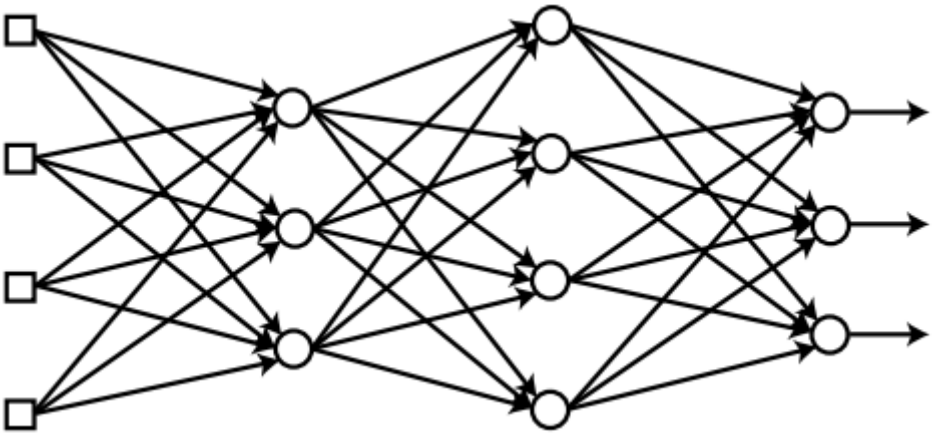


Figure 1.9: Réseau de neurones profond [2].

1.2.3. Les architectures les plus populaires des réseaux de neurones

Il existe des architectures qui sont très utilisés en Deep learning, parmi les plus populaires, nous avons l'algorithme de réseau de neurones convolutif CNN (**C**onvolutional **N**eural **N**etwork) et le réseau de neurones récurent **RNN** (**R**ecurrent **N**eural **N**etwork).

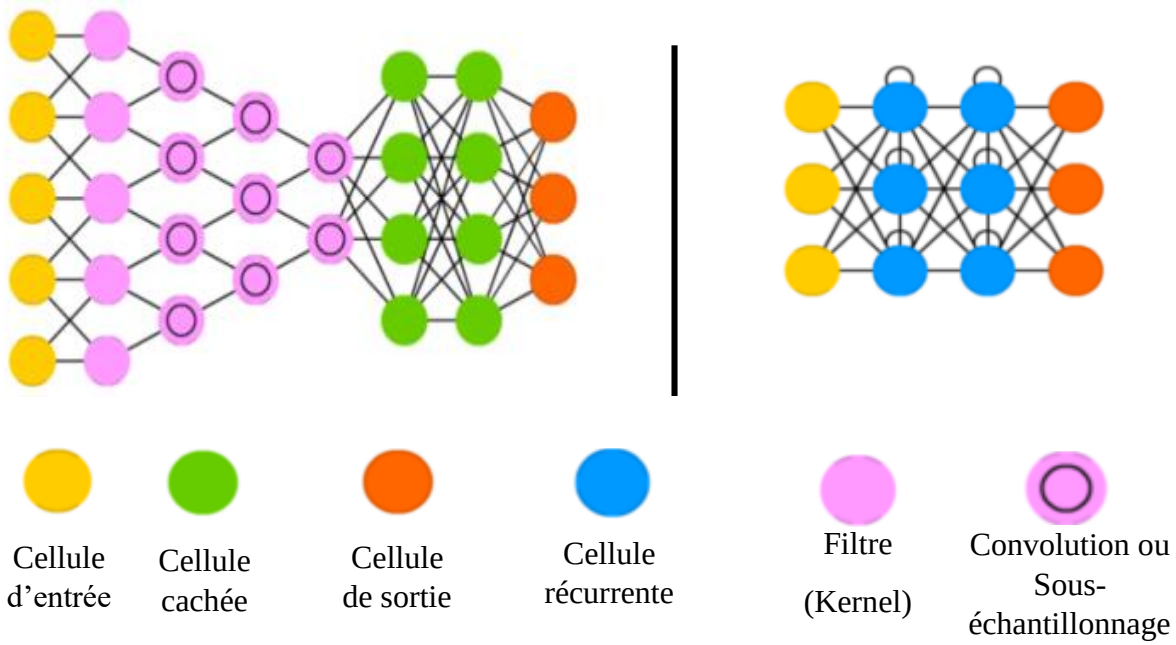


Figure 1.10: Architecture générale d'un réseau CNN (à gauche) et RNN (à droite) [12].

À partir de cette figure on peut dire qu'en fonction des besoins désirés, on peut implémenter une architecture à réseau de neurones comme dans notre cas par exemple, nous nous intéressons pour l'architecture CNN, car ils présentent beaucoup de performances surtout pour le cas de big data.

1.3. Vocabulaire des Réseaux Neuronaux

Nous avons présenté les généralités sur le Machine Learning afin d'introduire le Deep Learning. Cependant, il est primordial de connaître des concepts et vocabulaires de l'algorithme d'apprentissage pour une architecture d'un réseau de neurones, à travers ces vocabulaires, nous comprendrons ainsi la conception d'un apprentissage profond.

1.3.1. Fonction Coût (Cost Function)

Nous avons introduit l'erreur entre le modèle et les données. L'ensemble de ces erreurs est représenté par une fonction coût (En anglais **Cost ou loss function**). Cette fonction peut être définie avec plusieurs fonctions, l'une des plus fréquemment utilisée est l'erreur quadratique moyenne (Mean squared error), tel que :

$$E = \frac{1}{2m}\sum_{i=1}^{m}(Y_{ref} - Y)^2 \qquad (1.8)$$

Elle se traduit par la moyenne des erreurs $\boldsymbol{m}$ du nombre de neurones en sortie. $Yref$ change à chaque fois qu'il y'a une nouvelle mise à jour des paramètres [2][3].

En réseau de neurones, la fonction coût est exprimée par la technique de **Back propagation** [13].

1.3.2. La descente de Gradient

La descente de gradient (Gradient descent) représente l'algorithme d'apprentissage optimal, elle minimise la fonction coût pour prendre le résultat optimal possible et donc elle définit le processus d'apprentissage qui optimise les poids synaptiques et réduit le coût de la fonction [5]. La descente de gradient correspond aussi à (l'équation 1.3) où le processus avance par l'intermédiaire d'un pas d'apprentissage ; Ce sera un facteur pour l'avancement de la dérivée sur la fonction coût.

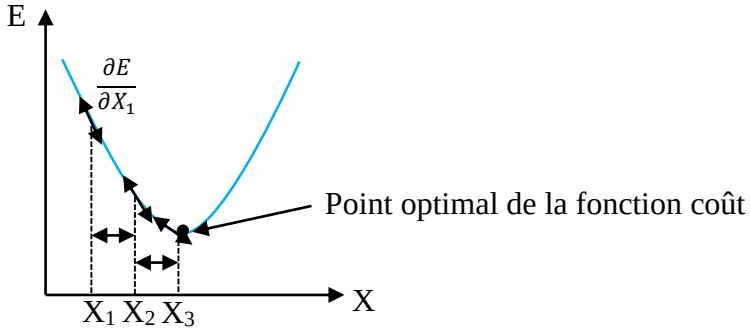


Figure 1.11: Optimisation de la fonction coût par descente de gradient.

Comme il est illustré sur cette figure ci-dessus, suivant des dérivées partielles qui correspondent à chaque erreur donc, l'ensemble de ces erreurs et par l'intermédiaire de la descente de gradient suivant un pas, nous avançons vers le point optimal, et si le pas et trop

grand nous risquons de le dépasser, et s'il est trop petit, nous risquons de ne jamais se rapprocher de ce point.

1.3.3. Pas d'apprentissage (Learning Rate)

Comme il a été cité dans la section précédente, c'est le pas d'apprentissage, il correspond dans l'équation (1.3) au coefficient $\boldsymbol{\alpha}$, il peut aussi exprimer la vitesse à laquelle la machine peut apprendre, cette vitesse influe sur le calcul des gradients pour optimiser la fonction coût. Cependant, il faut faire attention au sur-apprentissage (**Overfitting**) qui peut engendrer des dysfonctionnements avec des résultats de prédiction non corrects, car le fait de sur-apprendre conduit à mesurer aussi et modéliser le bruit contenu dans les données [7] (problème de divergence vers le point optimal).

Etant donné que notre étude est d'enlever le bruit des images, il est donc clair qu'il faut éviter ce phénomène d'overfitting.

1.3.4. Back-Propagation

En réseau de neurones, nous avons détaillé le principe de feed-forward la propagation avant, mais on peut se poser la question sur comment la mise à jour des paramètres W et B se fait dans le réseau. Une idée consiste à faire le chemin inverse, c'est-à-dire qu'après avoir obtenu la sortie Y avec une erreur, on calcule une chaîne de gradient à partir de la sortie par rapport à la dernière couche qui précède chacun ainsi de suite jusqu'à la première couche, ce principe est illustré dans la figure suivante où nous observons l'erreur en sortie qui se propage dans la couche qui précède la sortie selon comment elle varie. On obtient donc de nouveaux paramètres (W, B) mit à jour, avec $\boldsymbol{\varphi'}$ est la variation dans chaque neurone et υ est le produit de somme des paramètres dans le réseau.

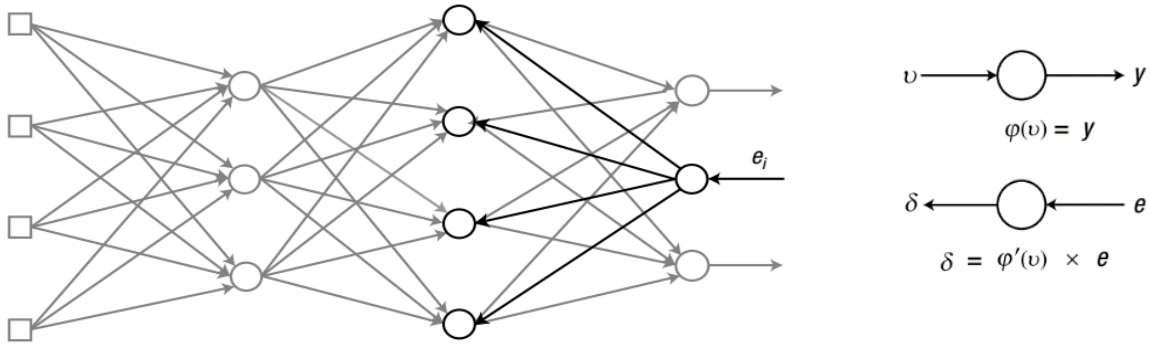


Figure 1.12 : Illustration du principe de back propagation [2].

1.3.5. Batches

Il est mentionné dans la figure 1.3 que dans le schéma d'un réseau de neurones profond il y'a des données d'apprentissage ; la notion de batch entre avec ces données que l'on va utiliser (Training Data) [8], l'importance des batches réside du fait qu'un lot de données (**samples**) est envoyé au réseau de neurones sous forme de mini-batches avec une taille spécifiée (batch size)[10]. Grâce à ceci, le modèle du réseau de neurones peut s'entraîner suivant le lot qu'il reçoit et à travers celui-ci il met à jours les paramètres [2]. La figure suivante montre un bloc de données à apprendre où on a un échantillon de données qui correspondent au batch :

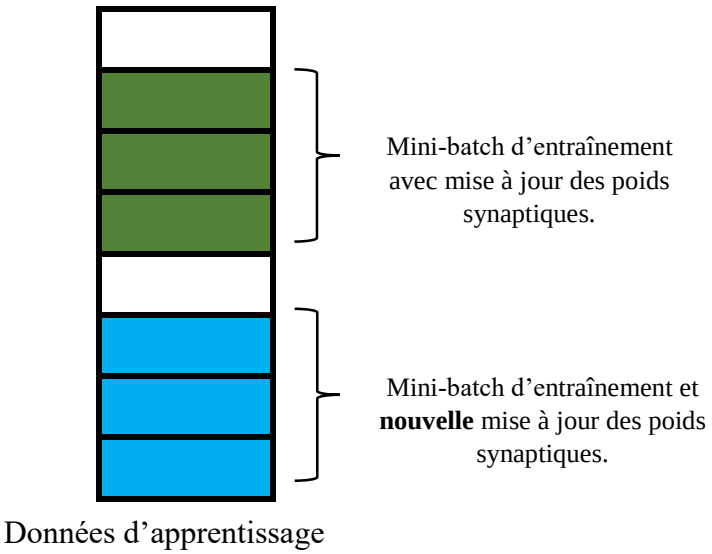


Figure 1.13: Exemple de mini-batch de données d'apprentissage

Cette notion de lots ou échantillons de données est très importante pour le prétraitement des données, cela aide à ce que le modèle parvient à mieux généraliser le phénomène avec qui il apprend pour avoir une prédiction adéquate et mieux adaptée [8]. Cela permet aussi de gagner en rapidité et d'éviter une très grosse accumulation d'erreur étant donné qu'un lot de données génère une fois la mise à jour des poids synaptiques.

1.3.6. Epochs

Le back-propagation et le feed-forward à eux deux représentent un epoch, dans un entraînement de réseau, cela représente le nombre d'itérations qu'exécute l'algorithme [6], ainsi que le nombre de passage des lots de données dans le réseau, cela peut aider à savoir quand

arrêter le processus d'apprentissage de façon critique [10] ; cela permettra en d'autres terme de valider l'apprentissage sur des données de test (test set).

Si nous avons par exemple 500 images en tant que training set et que nous choisissons de façon arbitraire 25 images qui seront le batch size (taille du lot), nous aurons alors la valeur de l'epoch qui sera de 20 ce qui donnera en face 20 fois le nombre de passages dans le réseau via la propagation avant et arrière et donc 20 updates effectués pour nos paramètres.

1.4. Réseaux de Neurones Convolutifs (Convolutional Neural Network CNN)

Rappelons pour mémoire que notre travail traite le débruitage d'images à base de réseau profond, il est donc judicieux de détailler l'architecture CNN. En fait, les CNN ont montré leurs performances en reconnaissance d'images [2]. Son principe repose sur le **Template Matching** [14] qui est très utilisé en vision artificielle et vision par ordinateur pour la reconnaissance des objets. Grâce à cela on peut extraire différentes informations utiles concernant l'image pour qu'au final, une classification d'images ou détection d'objets puisse aboutir [15]. Il faut savoir que l'image est une matrice remplie de pixels avec différentes intensités pour reproduire une apparence d'un objet ou un phénomène dans cette matrice [16]. Du coup le traitement se fera vis-à-vis de ces pixels afin de faire une extraction de caractéristiques (**features extraction)**. L'intérêt de cette extraction est la réduction du Dataset qui est immense, avec un nombre incalculable de paramètres. Si chaque pixel est connecté à un neurone en tant que donnée (X) d'entrée, il devient pratiquement irréalisable [10], c'est pourquoi avant d'entrer par un réseau feed-forward classique, les données passent par des couches d'extractions comme dans l'exemple de l'architecture de la figure 1.10 précédente, ou nous pouvons le démontrer de cette manière comme suit :

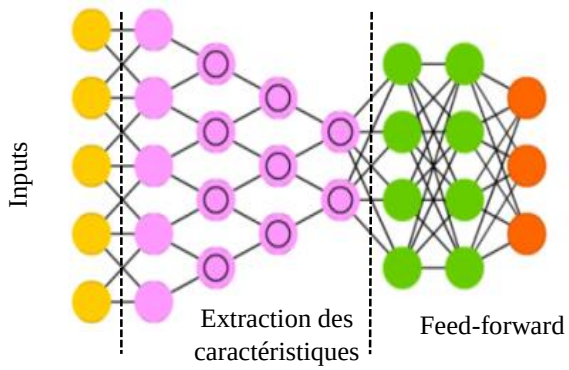


Figure 1.14: Architecture ConvNet composé d'un réseau spécial.

Contrairement au template matching manuel qui est indépendant du Machine Learning, en CNN l'algorithme d'apprentissage doit apprendre par lui-même l'extraction des caractéristiques de l'image [2], la figure suivante illustre de façon générale un réseau de CNN composé d'une extraction des caractéristiques où nous pouvons remarquer que le feature extraction entre au cours du processus d'apprentissage :

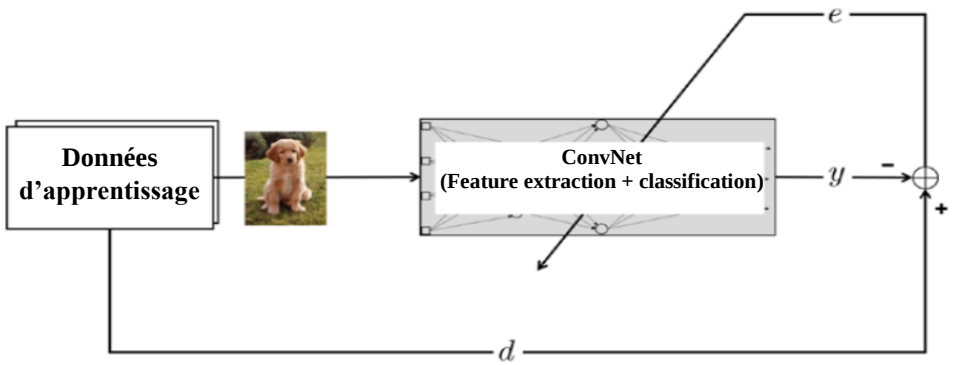


Figure 1.15 : CNN composé d'un réseau de neurones spécial.

Cette extraction se traduit en général par deux points fondamentaux qui sont la couche de convolution et la couche de pooling.

1.4.1. Couche de convolution

Afin de réduire le nombre de paramètres, il est parti du principe que des pixels voisins ont statistiquement plus de chance de correspondre à la même information par rapport à des pixels qui sont éloignés [14]. De ce principe la couche de convolution agit en tant que **filtre** aussi appelé **Kernel** de taille généralement de $2\times2, 3\times3, 5\times5$…etc, qui s'applique sur l'image d'entrée avant de passer dans un réseau feed-forward classique, cette taille de filtre va tout simplement convoluer avec une région de l'image d'entrée comme illustrée sur la figure suivante :

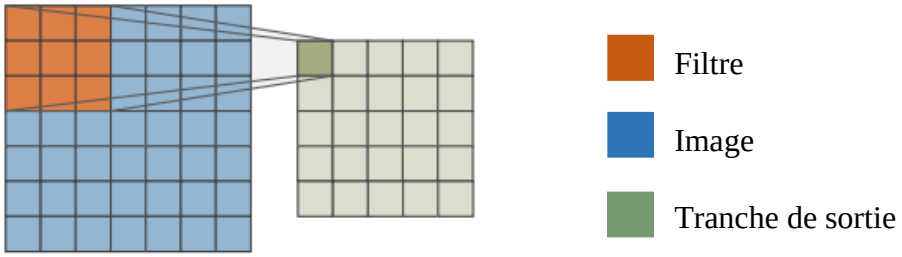


Figure 1.16: Convolution sur une Image d'entrée.

Le calcul de la convolution entre le filtre avec le bloc de l'image d'entrée (aussi appelé **receptive field**) va créer un bloc de sortie appelé **Feature map**, ce bloc de sortie résulte d'une convolution totale, c'est-à-dire qu'il passe par toutes les régions de l'image grâce au Kernel, c'est donc un balayage par le filtre qui est fait sur l'ensemble des blocs de pixels de taille ($M \times N$). Si nous avons une image d'entrée de taille ($H \times W$) [14], alors la taille du bloc qui résulte vaut :

$$H - M + 1 \times W - N + 1 \tag{1.9}$$

On peut le vérifier dans l'exemple de la figure 1.16 précédente ; où une image d'entrée de taille 7×7 convoluée avec un filtre de taille 3×3, le résultat est une couche de sortie de taille 5×5.

Remarque : Cette taille peut aussi correspondre au nombre de paramètres.

En réalité, le balayage apparaît comme une application de plusieurs filtres, mais avec des paramètres identiques de poids synaptiques propres au filtre ce qui est appelé **Weights sharing**. L'intérêt de cette technique de paramètres identiques à plusieurs filtres est pour éviter un grand nombre de paramètres qui sont calculés et palier au problème de sur apprentissage (**overfitting**) [11].

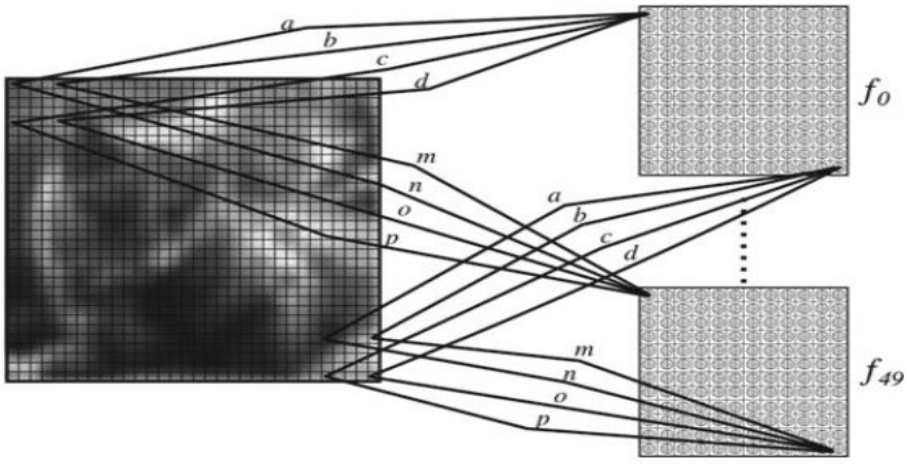


Figure 1.17 : Exemple de partage de poids synaptique dans le même bloc de neurones par rapport à d'autres blocs [14].

Le partage de poids peut s'illustré comme dans la figure ci-dessus où chaque neurone du bloc $\boldsymbol{fn}$ reçoit l'information du résultat du filtrage avec des paramètres (a, b, c, d) identiques pour l'ensemble du bloc avec un patch de taille 5×5 qui correspond au **receptive field**, mais

qui différent pour les blocs de neurones suivants (m, n, o, p). Pour cette raison, on dit que le bloc de neurone représente directement un seul filtre qui va en quelque sorte balayer et filtrer l'image d'entrée.

Ce que l'on vient de voir n'est autre qu'une seule extraction de caractéristiques. À partir de ce qui est illustré dans la figure 1.17 précédente, le partage de poids garanti une extraction de caractéristique bien spécifique ; pour des différents paramètres de filtres, on peut avoir plusieurs caractéristiques à extraire [11] comme pour le cas suivant :

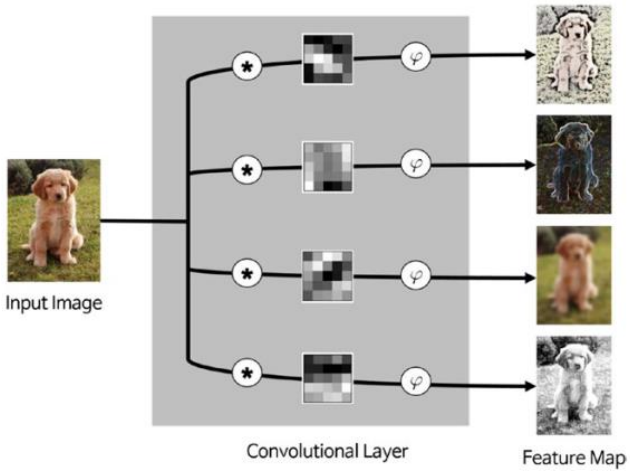


Figure 1.18 : Processus de Convolution avec plusieurs extractions de caractéristiques [2].

Le fait d'avoir plusieurs filtres convolutionnels acquiert plus de chance de développement d'apprentissage pour prédire les données, Par conséquent, chaque feature map résultante en sortie pourra alors passer par la fonction d'activation comme illustrée dans la figure 1.18 ci-dessus. Les filtres convolutifs $(*)$ appliqués à l'image d'entrée vont avoir en sortie plusieurs canaux de feature map, en passant par la fonction d'activation $(\boldsymbol{\varphi})$ de celles-ci on obtient différentes caractéristiques de l'image [2].

On ajoute aussi un point très important qui concerne les couches d'entrées qui doivent être d'une dimension $3D$ en raison que les images peuvent être en couleur RGB donc on ajoute une $3^{\text{ème}}$ dimension pour les filtres au lieu d'appliquer 3 filtres pour chaque canal (R, G, B), ce qui peut ressembler à cela (figure 1.19):

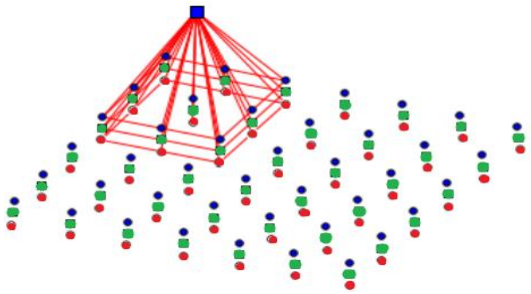


Figure 1.19 : Exemple d'un filtrage 3D avec 3 tranches d'images (Canal RGB).

La dimension de l'image d'entrée peut donc être de taille $W \times H \times C$, avec C qui représente le canal des trois couleurs, ou si dans le cas d'une image en niveau de gris elle devient la taille $W \times H \times 1$.

L'opération de convolution peut se traduire par :

$$\boldsymbol{T = X * W + B} \tag{1.10}$$

Où X est la matrice d'une région de l'image ; W la matrice des poids synaptiques ; B le vecteur des biais.

1.4.2. Couche de Sous-échantillonnage

Pour l'instant il n'est mentionné que des tailles d'images d'entrée qui sont relativement petites. Pour le cas d'images de taille 256 × 256 convolués avec 64 filtres de taille 9 × 9 par exemple on obtient 64 features de taille 248 × 248 avec comme résultats 3 936 256 paramètres à prendre en compte. Par conséquent, il existe une technique aussi efficace appelé **Pooling**. Cette technique permet de faire un sous-échantillonnage des blocs de sortie de la couche de convolution pour réduire la dimension des blocs des caractéristiques extraits soit en calculant la moyenne ou en prenant la valeur max suivant une taille de pooling spécifique (figure 1.20) :

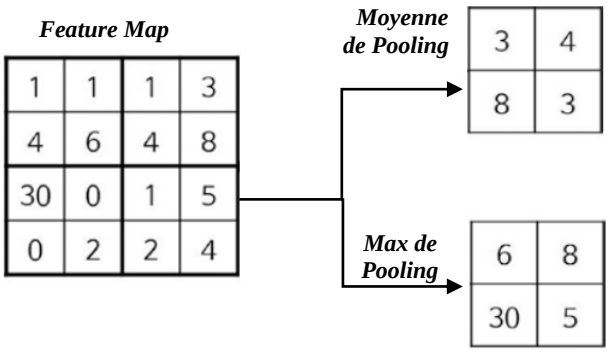


Figure 1.20 : Les résultats de deux méthodes de pooling.

De ce fait, les caractéristiques de l'image peuvent être conservées. En assemblant les deux couches de convolution et de pooling, on peut empiler plusieurs couches convolution/pooling une après l'autre et le principe reste toujours le même pour le calcul des caractéristiques des images. Par conséquent, pour garder les performances de classification intacte, il est préférable de prendre la première couche avec une taille assez grande, et au fur et à mesure que la couche se rétrécie au cours du passage à travers les couches suivantes [14].

1.4.3. Notion de stride et padding

On trouve des contraintes lors du filtrage convolutif, par rapport au positionnement des pixels et la taille des couches résultantes du filtrage :

- Il est remarqué que les pixels des bords de l'image sont peu exploités par rapport aux pixels centrés de l'image.
- La contrainte de réduction de l'image après le filtrage et Pooling induit à une limite par rapport aux nombres de couches convolutifs que l'on peut mettre dans le réseau neuronal.

Pour faire face à cela, deux techniques simples sont proposés. Faire un **Padding** pour augmenter la dimension des tranches d'images de sorties (figure 1.20) en ajoutant des zéros à l'extrémité de l'image d'entrée. Après le padding, l'image de sortie (output slice) aura la même dimension que celle de l'entrée, et les pixels de bords seront déplacés vers le centre et pourront mieux contribuer au filtrage convolutif comme on peut l'apercevoir sur la figure suivante :

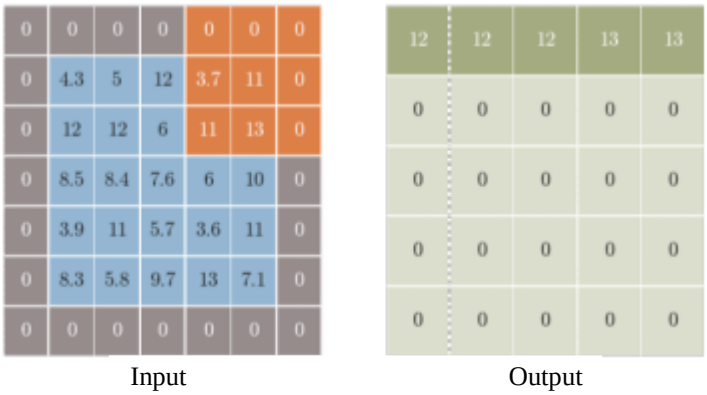


Figure 1.21 : filtrage d'image avec un kernel de 3 x 3, padding = 1 et stride = 1 [17].

En prenant P=1 le nombre de couche ajoutée à l'image avec la taille du kernel $(N \times M)$ comme pour le cas de cette figure ci-dessus, alors on aura la taille qui vaut $(H + 2P) \times (W + 2P)$, et dans ce cas la taille de feature map sera de :

$$(H + 2P - N + 1) \times (W + 2P - M + 1) \quad (1.11)$$

On trouvera donc que la taille de l'image résultante est la même que celle de l'entée, c'est-à-dire 5×5.

Le **stride** représente le pas du déplacement du filtre le long de l'image, par défaut celui-ci avance d'un pas, lorsque le stride augmente, le slice de sortie ou feature map de sortie rétrécie en termes de taille, mais cela permet de gagner plus en receptive field pour un neurone de sortie dans les couches qui vont succéder (Figure 1.22), et cela permettra de détecter plus largement des caractéristiques plus complexes [11]

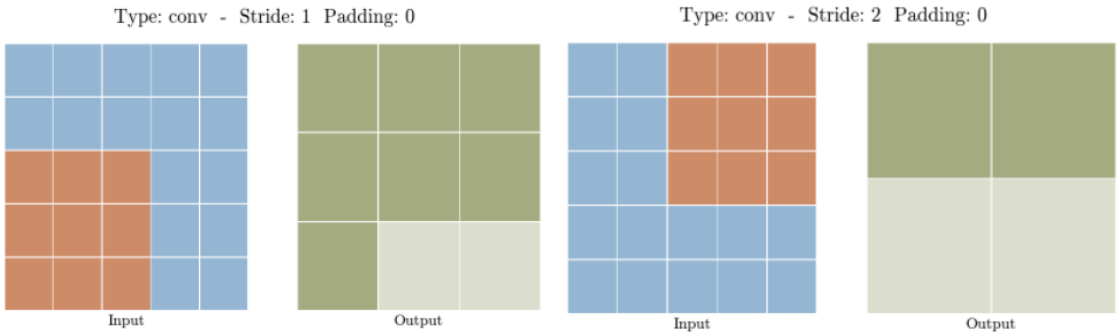


Figure 1.22 : Différence entre stride de 1 et stride de 2 [17].

Le résultat va produire une taille de :

$$\frac{H-N}{2} + 1 \times \frac{W-M}{2} + 1 \quad (1.12)$$

1.5. Auto-encodeur

Il convient de présenter l'auto-encodeur car il fera partie de notre implémentation des algorithmes de débruitage d'images et vidéos, étant donné que ce type d'algorithme peut être utilisé pour la suppression du bruit sur les données[5].

Un auto-encodeur est utilisé pour l'apprentissage non-supervisé où on cherche à rapprocher les données de l'entrée avec ceux de la sortie par correspondance et la similarité.

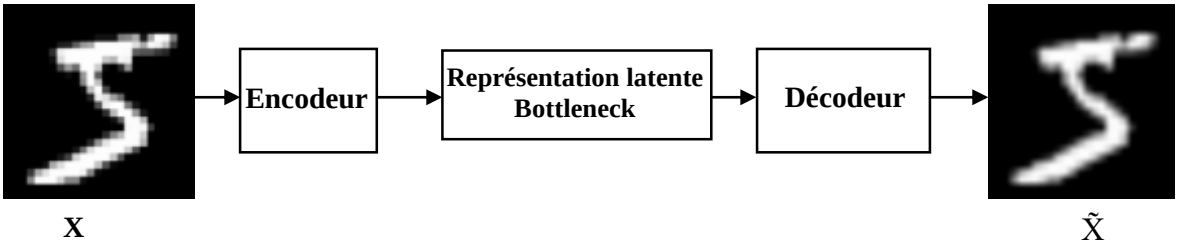


Figure 1.23: présentation de l'auto-encodeur.

L'architecture d'un Auto-encodeur passe par trois étapes, **l'encodage** $\boldsymbol{Z = f(X)}$ pour la compression de la donnée, **le décodage** $\boldsymbol{g(Z) = \tilde{X}}$, et une partie entre les deux qui est **l'espace latent** $\boldsymbol{Z}$ qui veut dire la partie cachée qui possède une petite dimension réduite et compressée par rapport à l'input (figure 1.23 avec les images extraites du dataset MNIST). La figure suivante illustre cette architecture d'auto-encodeur avec des neurones :

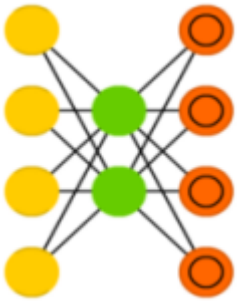


Figure 1.24 : Architecture simple d'un auto-encodeur [12].

Il y'a la couche d'entrée en jaune pour l'encodage, en vert la couche cachée pour les données compressées depuis l'entrée, et en sortie, la couche en orange pour la correspondance entre l'entrée et la sortie.

La couche bottleneck qui représente la partie cachée (Hidden layers), provient du principe de convolution et de là la donnée est compressée sous forme d'un vecteur, et c'est à partir de cette donnée que l'algorithme va devoir reproduire dans la sortie ce qu'il y'avait à l'entrée, c'est pourquoi l'algorithme d'apprentissage doit choisir minutieusement quelle importante caractéristique à prendre de ce vecteur [11].

Etant donné que l'on parle de correspondance entre l'entrée et la sortie, pour le cas du débruitage d'images, si nous avons à l'entrée des images bruitées, alors on aura un calcul de fonction coût qui peut être MSE qui est définie comme suit :

$$\mathcal{L}(x,\tilde{x}) = MSE = \frac{1}{m}\sum_{i=1}^{i=m}(x_i - \tilde{x}_i)^2 \quad (1.13)$$

Tel que x est l'image originale de taille m et $\tilde{x}$ l'image débruitée (estimée).

Ce qui revient alors à calculer le SSIM pour la similarité et aussi à mettre la fonction coût la plus petite possible afin de maximiser les chances de recouvrir la donnée initiale. Pour le débruitage, un bruit est appliqué à l'entrée afin de générer les données bruitées et qui sont les entrées du réseau [5]. La Figure 1.25 illustre les étapes de débruitage entre l'encodeur et le décodeur depuis la génération des données bruitées.

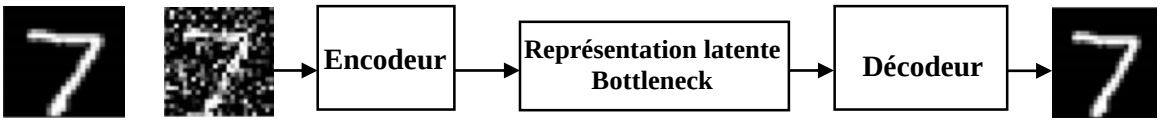


Figure 1.25 : Débruitage Auto-encodeur en ajoutant de bruit à l'entrée.

1.6. Conclusion

Nous avons présenté à travers ce chapitre les fondamentaux du deep learning à travers les principaux points majeurs comme **l'importance du dataset, la fonction coût** à travers le modèle développé par l'algorithme d'apprentissage ainsi que **l'optimisation par gradient de celui-ci** en parallèle avec **le back-propagation** lorsqu'on parle de réseau de neurones**.** Ces outils qui entrent dans la conception d'un réseau de neurones artificiel multi-couches et plus précisément le CNN pour notre cas qui sera un réseau clé pour les chapitres qui vont suivre à travers le choix des filtres, et l'importance des paramètres à initialiser et optimiser pour avoir de bonnes performances. Nous avons aussi fait une brève introduction sur l'auto-encodeur en vue du travail que l'on veut faire pour le débruitage d'images et potentiellement des séquences vidéo.

Chapitre 2

Débruitage Image & Vidéo

Résumé

Ce chapitre traite le débruitage images/vidéos, et présenter les méthodes et les algorithmes utilisés récemment. Nous présenterons les principes de débruitage à base de réseaux de neurones qui s'ajoutent dans les architectures et qui ont pu apporter des solutions à certains problèmes.

Sommaire

2.1. Introduction

Dans notre domaine de la science des technologies, et notamment en électronique des systèmes embarqués en général, le bruit est un phénomène perturbateur qui ne peut être décri ou modélisé de manière déterministe. Il peut s'ajouter sur des appareils électroniques lors des mesures comme le cas des capteurs photographie et toutes sortes de systèmes d'acquisitions de données [18]. On doit étudier comment se familiariser avec ce phénomène (signal supposé aléatoire) qui s'ajoute sur nos données (images/vidéos), donc différentes techniques que ce soit classiques ou via les algorithmes par réseau de neurones se proposent qui sont : **Block Matching 3*D* (BM3D), Descriminative Learning based method (DnCNN), FFDnet** pour le cas des images, et **SPTWO, Video Non-Local Bayes (VNLB), Video Block Matching 4D (VBM4D), Deep Video Denoising Network (DVDnet), Fast Deep Video Denoising Network (FastDVDnet)** pour les vidéos. Où chacun possède ses propriétés spécifiques que ce soit débruitage d'image et/ou vidéo.

2.2. Algorithmes de Débruitage d'Images

Le débruitage aussi appelé lissage peut être effectué par de différente manière, la technique la plus connue étant par le filtrage. Il existe plusieurs types de filtres (selon le type de bruit à atténuer), comme les filtres linéaires, ils sont nommés ainsi car ils vérifient la propriété d'additivité et d'homogénéité. Le plus utilisé est **le filtre Moyenneur**. Le principe du filtre moyenneur réside dans le calcul de la moyenne sur les pixels en fonction des pixels voisins de celui-ci pour donner une sorte de cohérence entre les pixels [18], la figure suivante illustre les pixels sous forme d'un tableau ou on utilise une opération de lissage par moyennage :

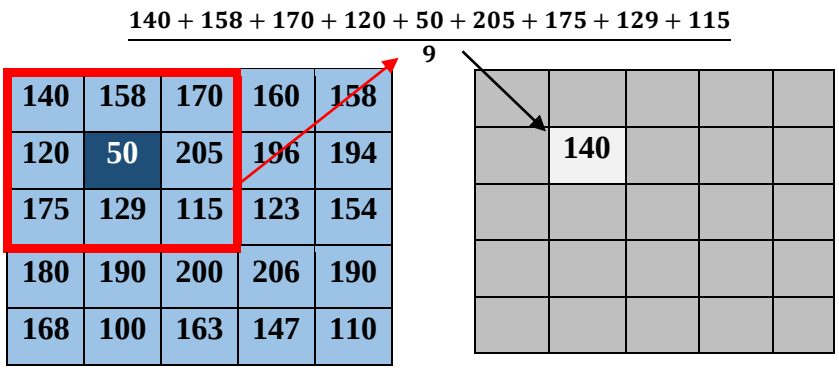


Figure 2. 1 : Exemple d'opération de filtrage par moyenneur.

Il s'agit donc d'une opération de convolution $T = H * I$, à noter que $*$ représente l'opération de convolution suivante :

$$T_{i,j} = \frac{1}{D}\sum_{u=-k}^{k}\sum_{v=-k}^{k} H_{u,v}.\, I_{i-u,j-v} \qquad (2.\ 1)$$

Où $\boldsymbol{H}$: filtre ou noyau, aussi appelé masque de convolution (matrice carrée de taille impaire). Et $\boldsymbol{D}$: coefficient de normalisation $D = 2k + 1$ en fonction du filtre impair. $\boldsymbol{I}(X, Y)$: la taille de l'image originale. Ainsi on obtient l'image $\boldsymbol{T}$ filtrée de taille $(X \times Y)$.

Un autre filtre linéaire connu et très utilisé et qui nous intéresse le plus qui est **le filtre gaussien**, il est représenté sous forme d'une distribution normale selon la fonction définie comme suit :

$$G(X, Y) = \frac{1}{2\pi\sigma^2} exp\left(-\frac{X^2+Y^2}{2\sigma^2}\right) \qquad (2.\ 2)$$

Ce filtre G(X, Y) utilise un noyau différent de celui du filtre moyenneur, le noyau est gaussien (équation 2.2) avec $\boldsymbol{\sigma}$ **:** représente **l'écart type**, X et Y : coordonnées de l'origine de l'image. La forme de cette fonction est montrée sur la figure 2.2 :

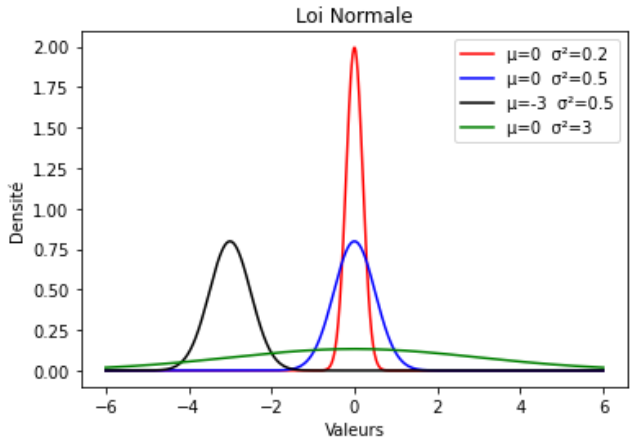


Figure 2.2 : Exemples de distribution gaussienne.

Nous avons selon la valeur de la variance $\boldsymbol{\sigma}^2$ l'allure de la gaussienne. Elle joue sur l'ouverture de la fenêtre du filtre et la valeur de la moyenne aussi appelée espérance de la distribution $\boldsymbol{\mu}$ qui agit comme un offset sur le filtre. Certes, l'image filtrée aura un bruit atténué, mais en revanche celle-ci se verra un peu floutée selon l'augmentation de la variance ou la moyenne qui vont affecter le contraste de l'image [18].

Dans un souci de classification, il convient de définir d'abord le bruit. **Le bruit blanc additif gaussien** est défini en tant que distribution aléatoire avec des variables X_i aléatoire non déterministes distribués en fonction de la moyenne (équation 2.3) et de la variance (équation 2.4) :

$$\hat{\mu} = \frac{1}{N}\sum_{i=0}^{N} X_i \tag{2. 3}$$

$$\hat{\sigma}^2 = \frac{1}{N}\sum_{i=0}^{N}(X_i - \hat{\mu})^2 \tag{2. 4}$$

Cela représente plusieurs paramètres de la moyenne et de la variance qui vont affecter chaque pixel qui sera bruité, ce qui est difficile dans ce cas, c'est le fait de traiter séparément chaque source de bruit [19]. Par conséquent, on procède par le bruit gaussien additif suivant :

$$s = \sqrt{\sum_{i=1}^{N} {\sigma_i}^2} \qquad \frac{1}{s}\sum_{i=1}^{N}(X_i - \mu_i) \xrightarrow{d} \mathcal{N}(0,1) \tag{2. 5}$$

Représentant une distribution identique pour les paramètres ${\sigma^2}_i$ et μ_i. $\xrightarrow{d}$ désigne la convergence de la distribution, et $\mathcal{N}(0,1)$ indique respectivement la moyenne et la variance égale au premier et deuxième argument, ce qui devient un paramètre par défaut par la suite. Le bruit gaussien étant assumé additif, alors l'équation de l'image résultante par rapport à l'image originale avec le bruit donne :

$$Z(X) = Y(X) + \eta(X) \tag{2. 6}$$

Donc pour récapituler, une image Z capturé selon x la coordonnée du pixel, nous avons l'image y qui est l'image inconnu (à estimer) en bonne qualité, η représente le bruit additif selon $N\ (\mu, \sigma^2)$[20].

Nous pouvons aussi ajouter d'autres bruits comme le bruit **poivre et sel**, **le bruit multiplicateur** et bien d'autres bruits. Pour mieux voir l'effet du bruit sur les images, nous avons simulé et généré des images bruitées contaminées par différents types de bruit. Sur la figure ci-dessous nous pouvons voir les types de bruits appliqués à une image :

a) Image originale.

b) Bruit Gaussien.

b) Bruit poivre et sel

d) Bruit multiplicateur

Figure 2.3: Images bruitées par différents types de bruit.

L'importance du filtre ne réside pas quand pour l'atténuation du bruit, mais il sert aussi à d'autres techniques de traitement d'image comme la détection des bords pour différencier entre les objets dans une image [21]. Le processus qui permet de grouper plusieurs zones ou régions d'une image est la **Segmentation**, en utilisant certains critères pour distinguer ces zones [18]. Deux points principaux en segmentation sont : la **valeur de similarité** et la **proximité spatiale**. Cela veut dire que deux pixels peuvent appartenir à la même région s'ils ont les mêmes caractéristiques d'intensité de pixel ou s'ils sont proches l'un par rapport à l'autre [18]. C'est pourquoi, dans la suite de ce chapitre, on trouve dans les algorithmes de débruitage que ces principes sont utilisés.

- **Block Matching 3D (BM3D)**

Le débruitage par algorithme BM3D se concentre sur les correctifs d'images seulement, avec la technique par patchs. Cet algorithme très connu, avec une utilisation sans réseau de neurones. Le but étant de faire une comparaison de celui-ci avec les autres algorithmes de CNN et constater les résultats obtenus.

Le principe de cet algorithme repose sur un processus de filtrage collaboratif, c'est-à-dire qu'à partir d'une image, on extrait plusieurs fragments de blocs qui sont similaires par rapport

à un bloc de référence à proximité appelé « R » (la similarité peut aussi être en termes de bruit appliqué sur les blocs).

Ces fragments de blocs d'images $2D$, seront représentées dans un tableau de données 3D, si les fragments de patchs sont de même taille, ils peuvent être représentés sous forme cylindrique. Le filtrage collaboratif entre en jeu pour traiter ce groupement de blocs en $3D$ [22].

Remarque : La notion de blocs, peut être interprétée comme des patchs.

La Figure 2.4 illustre le principe de l'algorithme BM3D.

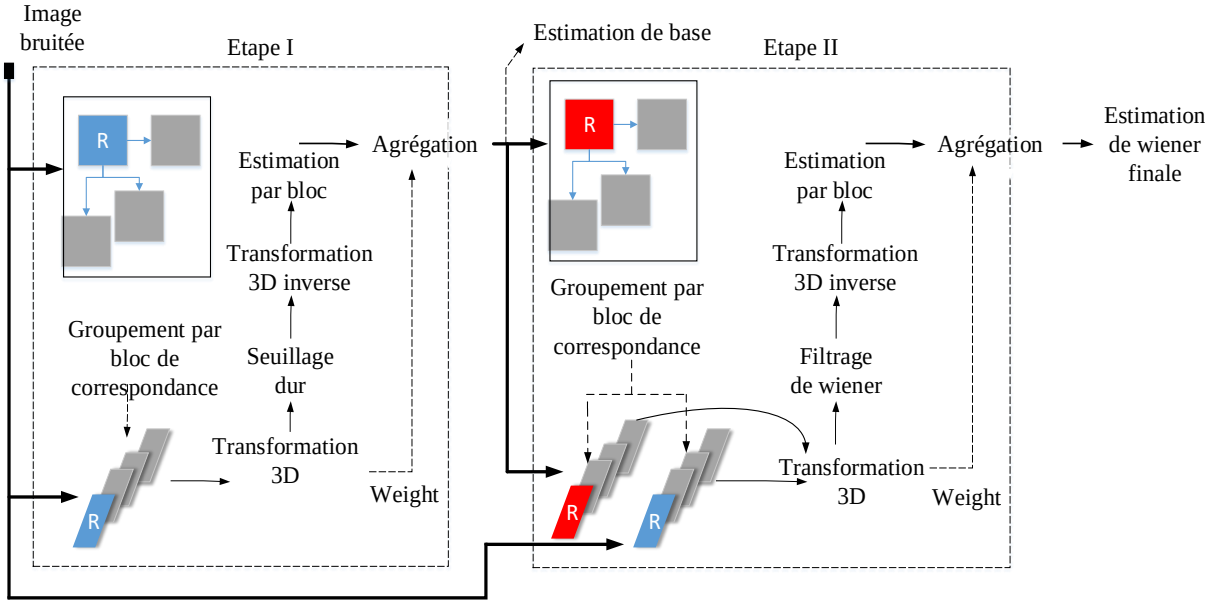


Figure 2.4 : Schéma descriptif du principe de la correspondance par blocs de l'algorithme BM3D.

Le principe fondamental est illustré sur le schéma de cette figure où le débruitage passe par deux étapes identiques à la seule différence, dans la première étape on trouve un seuillage dur et dans la 2ème étape on a un filtrage de Wiener, en prenant en compte que la 2ème étape est entamée en fonction des résultats de la 1ère [22].

Dans l'idée de blocs similaires, on peut ajouter un paramètre qui entre concernant les blocs par rapport à un patch de référence où la similarité entre des fragments de signaux est calculée selon une distance qui implique que plus les fragments sont mutuellement proches plus il y ait une correspondance. Cependant, lorsqu'on a une incertitude entre de correspondance

accompagné d'un bruit, il est nécessaire de passer par une extraction de caractéristiques du signal en premier lieu ensuite le calcul de la distance de ces caractéristiques [22].

- **Descriminative Learning based method (DnCNN)**

Nous présentons dans cette section la catégorie des algorithmes de débruitage à base de réseaux de neurones convolutifs. Parmi les algorithmes que nous avons étudié et implémenté, l'algorithme DnCNN. Il contient une architecture à base de réseaux de neurones convolutifs CNN [23] et qui se base sur l'apprentissage discriminatif fondé sur l'apprentissage résiduel (Residual Learning) et la normalisation de Batch (Batch Normalisation) dont peuvent tirés l'un à l'autre dans le réseau de neurones CNN. L'apprentissage résiduel permet de régler la précision (Accuracy) afin d'améliorer la prédiction de part un apprentissage ou entraînement qui devient plus facile lorsque le réseau est plus profond avec plus de couches cachées (Hidden Layers) on observe plus précisément le réseau résiduel avec des fonctions résiduelles référencées par rapport aux couches d'entrée [24],

La normalisation de Batch quant à elle, donne un rendement plus rapide qui est assurée par l'utilisation de mini-batchs ou mini-lots de descente de gradient stochastique (SGD). Son intérêt réside dans le fait que la distribution de chaque entrée de couches change lors de l'apprentissage et qui crée le phénomène de décalage de co-variable interne (internal covariate shift), induisant à plusieurs paramètres à connaître et faire initialiser de façon adéquate et précise (Il en est de même pour le taux d'apprentissage). Grâce à la normalisation des couches d'entrées, on peut pallier à ce phénomène, et l'apprentissage devient plus simple. Dans certains cas, cette normalisation peut éliminer le décrochage (Dropout), et elle agit comme une régularisation, c'est pourquoi celui-ci sera utiliser pour chaque mini-batch entraîné [25].

Ces deux concepts de normalisation et de réseau résiduel donnent alors un bienfait renforçant les performances de débruitage de notre réseau CNN. De plus, à l'aide des processeurs graphiques (GPU) modernes et puissants que l'algorithme DnCNN peut utiliser, nous pouvons atteindre des temps d'exécutions plus performants [23].

- **FFDnet**

L'algorithme FFDnet [26] est une autre technique utilisée pour débruiter les images en Deep Learning grâce au deep residual learning comme pour le cas du DnCNN, où on trouve

l'apprentissage résiduel et la normalisation par batch qui entrent dans les couches cachées profondes.

Cependant, la différence entre cet algorithme et celui mentionné précédemment est qu'il inclut des couches de **prétraitement** (preprocessing) et posttraitement (postprocessing) qui viennent respectivement avant et après la cartographie effectuée par DnCNN. En d'autres termes, l'image concernée passe par une réduction des pixels de l'image du quart de sa résolution en image multi canal (sous-images), incluant avec lui la carte du bruit (morceau bruité) [26]. Grâce à cette modification, il s'ajoute un nouvel aspect à l'architecture, de façon globale, la complexité de cet algorithme est réduit avec cette méthode. Le posttraitement permet de reformer et récupérer l'image initiale avec réduction du bruit. En comparaison par rapport à l'algorithme prédécesseur du FFDnet, celui-ci utilise un champ réceptif (receptive field) plus large grâce à la réduction de la résolution lors du prétraitement. Aussi, contrairement à l'algorithme DnCNN, l'image de sortie n'est pas une image à bruit résiduel comme dans la figure 2.5, mais une image latente [26].

Dans la phase du préprocessing, les différents canaux sont représentés selon la figure qui suit où la cartographie de la partie bruitée est inclue dans les multicanaux à résolution réduite :

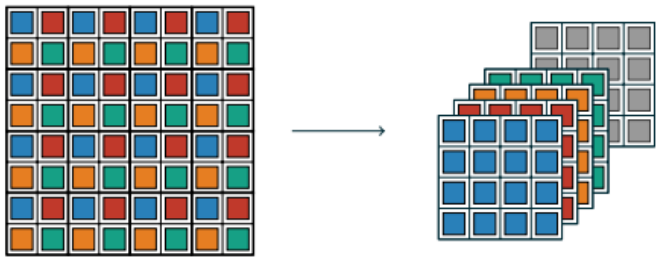


Figure 2.5: Diagramme de réduction de la résolution (Dawnscaling) [26].

Donc, on a une séparation des canaux en plusieurs sous-images avec réorganisation des pixels. Pour le cas opposé, c'est-à-dire le posttraitement où comme on peut voir l'exemple de la figure 2.6 avec une restitution des pixels et de la résolution :

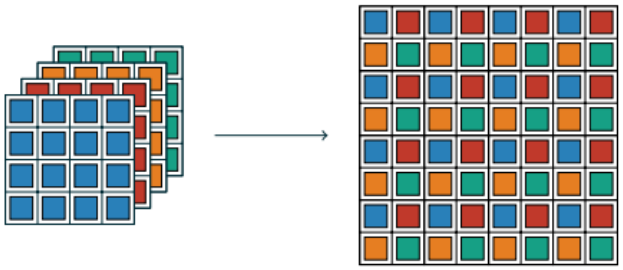


Figure 2.6: Diagramme de restitution de la résolution complète (Upscaling) [26].

Entre les deux phases (preprocessing et postprocessing), il y'a la phase de mappage non-linéaire composé de couches convolutifs les mêmes que ceux utilisés pour le cas du DnCNN avec comme filtre représenté par une matrice carrée 3×3 qui s'applique aux sous-images séparées pour extraire les caractéristiques (feature map) de la même taille que les sous-images lorsque le stride vaut 1.

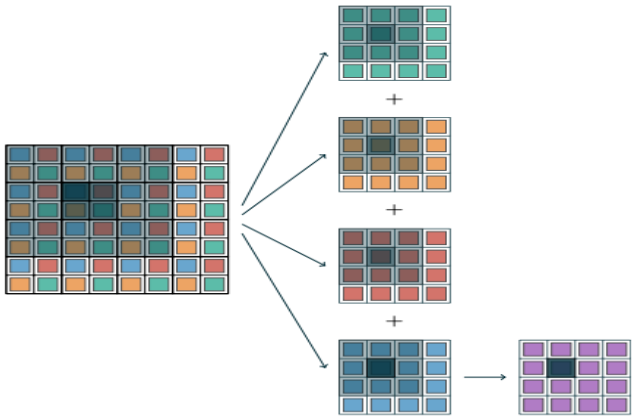


Figure 2. 7 : Extraction des caractéristiques depuis les images multi-canaux [26].

Les sous-images étendus grâce à un patch de 2×2 à partir de l'image originale balayé sur une fenêtre de taille 6×6 (figure 2.7) qui sont couvert en gris.

2.3. Algorithme de Débruitage Vidéo

Une vidéo est une séquence d'images aussi appelée séquence de trames donnant une sensation que ces images sont en mouvement [27]. Le principe d'une vidéo est un défilement ou balayage numérique des pixels de façon uniforme entre un pixel de la trame T avec le pixel de la trame $T - 1$ et $T + 1$ [23]. La complexité des vidéos par rapport aux images est le fait de l'apparition de la cohérence temporelle qui est cruciale et doit être stable. L'idée avec les images dans les sections qui ont précédé est que celle-ci reste la même, à la seule différence, il faut

étendre les régions avoisinantes vers les trames adjacentes. Plusieurs algorithmes de débruitage vidéo sont dédiés.

- **Algorithme Spatiotemporal With Optical Flow estimation SPTWO**

L'algorithme SPTWO est une structure sans réseau neuronal mais qui nous intéresse en termes d'étude comparative avec d'autres algorithmes qui comprennent la structure CNN pour le débruitage vidéo.

Le débruitage par SPTWO qui est un débruitage par estimation du mouvement par flux optique, consiste en une estimation de la trajectoire à partir d'une **trame de référence** vers une **trame cible** [28], sachant qu'il n'est pas obligatoire que la trame cible vient après la trame de référence. Grâce à l'invariance d'intensité des pixels on peut estimer la trajectoire qui est la suivante :

$$I(x, y, t) = I(x + u, y + v, t + 1) \quad (2.7)$$

Où $I(X, Y, T)$ est l'intensité du pixel dans la localisation (X, Y) sur la trame T, u et v sont le mouvement horizontal et vertical sur une position spatio-temporelle d'un pixel spécifique. Dans le cas où nous avons un bruit additif, dans ce cas l'équation devient :

$$I(X, Y, T) = I_0(X + u, Y + v, T + 1) + n(X + u, Y + v, T + 1) \quad (2.8)$$

Le calcul du flux optique s'effectue dans l'algorithme lors de l'alignement des séquences d'images. Grâce à celui-ci, on peut faciliter le voisinage entre les trames I_k des pixels cibles, suivi de l'interpolation bicubique afin d'avoir à la fin le calcul du flux optique, une correspondance de l'ensemble des voisinages dans un espace spécifique au pixel donné. Cependant, étant donné que nous avons le bruit blanc additif gaussien qui s'ajoute aux trames, alors l'alignement du nombre de trames correspondantes peut être réduit en raison du bruit [29]. C'est pourquoi, une détection d'occlusion qui dépend de la divergence et de la couleur des trames est effectuée.

Il faut aussi savoir qu'une sélection de patches similaires des trames alignées temporellement dont le débruitage est effectué dessus en blocs de patches rassemblés en $3D$ et que pour chaque patch similaire (comme pour le cas du BM3D) un débruitage est effectué prenant en considération la distance spatiale de ces derniers. À partir des blocs de patches $3D$, il existe M

$2D$ patches d'images de taille d'une matrice carrée $n \times n$ qui contiennent l'occlusion en question qui passent donc par le débruitage en l'utilisant avec la technique d'analyse en composante principale (PCA).

- **Video Block Matching and 4D filtering (VBM4D)**

L'algorithme VBM4D est une modification de l'algorithme BM3D. Son intérêt réside dans le groupement spatio-temporel et pas que des blocs de patches, mais plutôt des trames constituant un volume $3D$ défini par un vecteur de mouvements (motion vectors).Ces volumes similaires sont groupés avec le même principe que celui cité dans l'algorithme BM3D, sauf que dans ce-cas là on aura une dimension de plus, du fait d'être une structure $4D$. À l'intérieur de cette dimension $4D$ on trouve plusieurs données de corrélation c'est-à-dire qu'on trouve **la correlation locale** entre les blocs $2D$, **la correlation temporelle** entre la trajectoire du mouvement de la scène, et **la correlation spatiale non locale** entre les pixels.

On extrait à partir de la vidéo bruitée un volume spatio-temporel qui représente une séquence de blocs $3D$ construite suivant une trajectoire bien spécifique dans le temps, qui est supposé suivre le mouvement de la scène [30].

- **Video Non-Local Bayes (VNLB)**

Un autre algorithme connu pour le débruitage vidéo qui est une extension d'un algorithme de débruitage d'images est l'algorithme NLB. Il se base sur l'estimation bayesienne, non locale qui veut dire que les positions de pixel à travers l'image peuvent être dans des régions différentes de l'image. C'est aussi une extension de l'algorithme de l'estimation moyenne des pixels similaires de l'image NL-means[31]. La structure de l'algorithme peut se rapprocher de l'algorithme BM3D étant donné que celui-ci possède aussi deux étapes d'estimation pour le débruitage [32]. C'est pourquoi nous avons choisi d'en parler que sur l'algorithme BM3D précédemment dans la section des images et rassembler les notions de l'algorithme par estimation bayésienne dans cette section, ajouté à cela bien sur l'aspect spatio-temporel dans la vidéo. L'estimation se base sur la loi de bayes qui est la suivante :

$$P(A|B) = \frac{P(B|A)(vraissamblance) \times P(A)(distribution\ a\ priori)}{P(B)évidence} \quad (2.9)$$

La méthode est construite par un modèle bayésien pour chaque groupe de patches spatio-temporel sans tenir compte de la compensation du mouvement ce qui permet d'enlever l'erreur causée par l'estimation du mouvement (motion estimation) [33], se basant beaucoup plus sur la moyenne pondérée des patches similaires groupés en fonction des voisins les plus proches sur lesquels l'estimation et le débruitage seront centré dessus.

L'avantage dans le spatio-temporel, c'est de pouvoir utiliser les patches $3D$ (qui ont été proposés dans cet algorithme). On a deux types de patches, un patch rectangulaire, et un autre patch à mouvement compensé (motion compensated). Le patch dans ce cas peut exploiter de manière adjacente les trames $2D$ qui sont dans les tranches temporelles c'est-à-dire à partir du groupe $3D$ obtenu.

L'avantage de cet algorithme réside dans le fait de proposer un débruitage par patch plus puissante avec deux cas possible, cependant la charge de calcule est grande prenant ainsi un temps d'exécution assez long.

- **Deep Video Denoising (DVDnet)**

Nous traitons maintenant à la partie des algorithmes très récents et qui nous intéressent le plus en termes de débruitage vidéo. L'originalité de la technique de débruitage par l'algorithme DVDnet réside dans l'utilisation d'un réseau neuronal CNN.

Deux principales parties vont composer cet algorithme, à savoir le **débruitage spatial**, et le **débruitage temporel** incluant aussi la compensation du mouvement [34]. Nous pouvons montrer cela à partir de l'architecture de la figure suivante :

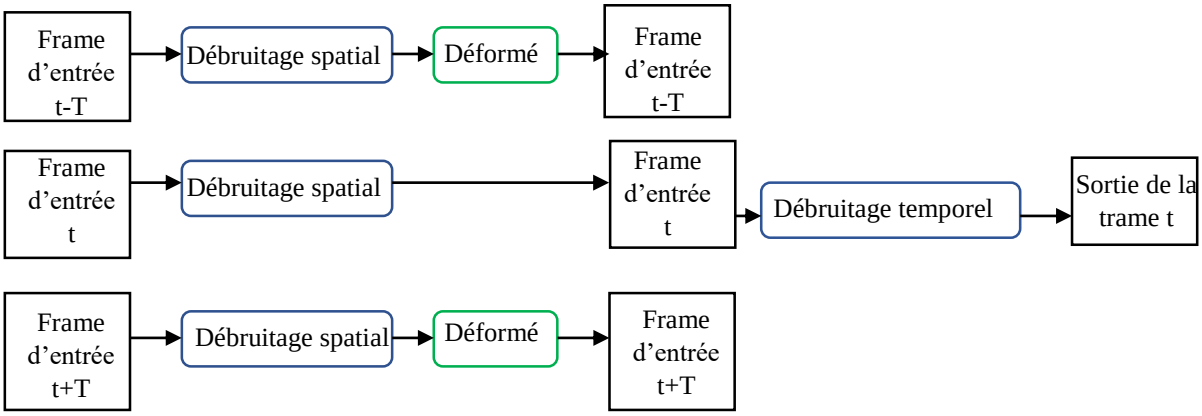


Figure 2. 8 : Architecture simplifié du débruitage par DVDnet.

Les $2T$ trames voisines représentant l'entrée du réseau passeront par le débruitage spatial, cependant même si le débruitage s'effectuera de façon adéquate avec les pixels des trames mais il y'aura un problème de cohérence temporel c'est pourquoi un alignement vers la trame centrale est effectué constituant une nouvelle entrée au débruitage temporel tout en estimant le flux optique (Figure 2.8).

- **Fast Deep Video Denoising (Fast DVDnet)**

L'algorithme FastDVDnet est une version modifiée de celui précédemment mentionné, la modification apportée au FastDVDnet se concentre sur la suppression du flux optique. À la base les algorithmes considérés dans cette étude estiment le flux optique pour compenser le mouvement mais cela coûte du temps pendant le **running time** du fait de son nom FastDVDnet. Avec cette suppression l'algorithme gagne en rapidité de débruitage.

L'entrée de l'algorithme comprend 5 frames avant le passage vers les blocs de débruitage qui eux vont recevoir chacun trois frames ensuite, un passage vers un deuxième bloc qui est similaire, comme nous pouvons le voir sur le schéma suivant :

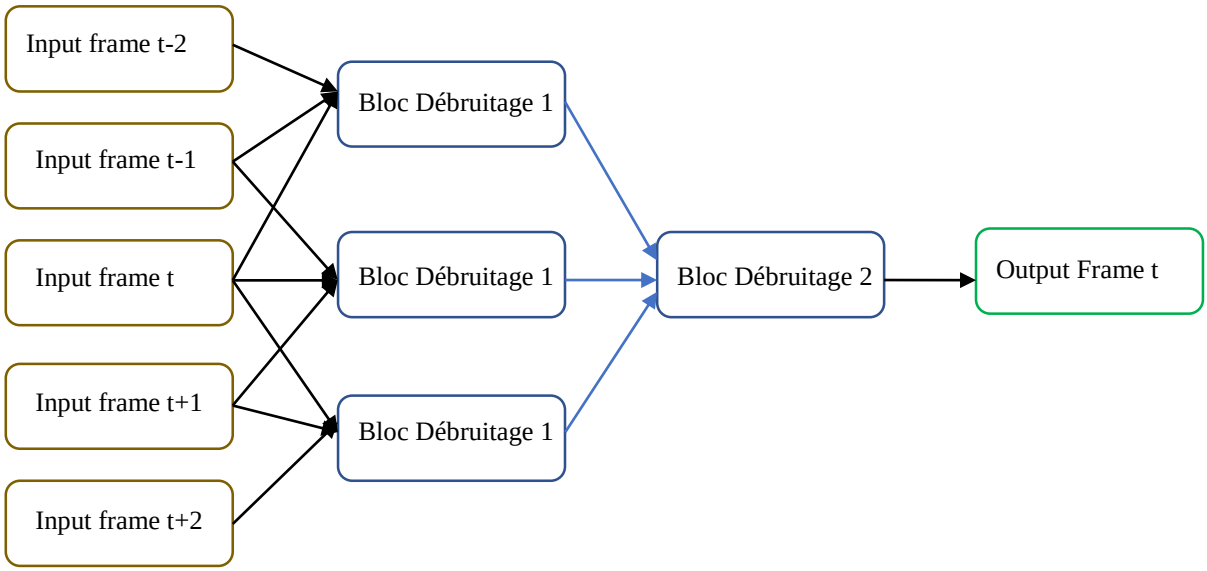


Figure 2. 9 : Blocs de débruitage 1 et 2 en cascade avec triplé de trames d'entrée.

Cette architecture en cascade permet de renforcer avec une application d'une corrélation temporelle du bruit restant dans les trames de sortie [35]. Chaque bloc prend un triplé des trames

à l'entrée et leur sortie devient l'entrée du deuxième bloc, la sortie de ce dernier devient une estimation de la trame centrale [31].

2.3. Conclusion

Nous avons pu voir dans ce chapitre les différentes techniques adoptées dans l'état de l'art du débruitage image et vidéo. Plusieurs algorithmes possèdent des techniques en commun où comme la débruitage par Patch qui est très utilisé et exploité que ce soit pour les images ou pour les vidéos, pour le cas des vidéos il est primordial de prendre en considération la complexité de la cohérence temporelle. Chacun des algorithmes prend un avantage plus que l'autre dans certains cas par rapport à d'autres, et bien sur l'un peut dépasser l'autre en termes de rapidité. C'est pourquoi dans le prochain chapitre nous aborderons l'implémentation de ces algorithmes et voir ce que donne les résultats en terme de qualité de débruitage.

Chapitre 3

Implémentation des algorithmes et résultats des expériences

Résumé

À partir de ce que nous avons cité dans le chapitre 1 et 2, nous allons implémenter les algorithmes de débruitage suscités ainsi qu'une structure CNN afin de montrer l'intérêt des CNN. Nous utiliserons plusieurs environnements ainsi que des logiciels spécifiques pour traiter les données avant le débruitage surtout pour le cas des séquences vidéo.

Sommaire

3.1. Introduction

À travers la partie théorique sur le **Deep Learning** et les réseaux de neurones convolutionnels CNN et le débruitage des images et vidéo, nous présenterons dans cette dernière partie les différents résultats et testes des expériences utilisées sur des algorithmes des structures de modèle implémenté avec aussi ceux que nous avons cité dans la partie théorique du chapitre 2 notamment: **BM3D**, **DnCNN**, **FFDnet, SPTWO**, **VBM4D**, **DVDnet**, **FastDVDnet**, **VNLB**. Pour établir une étude comparative, quantitative et qualitative entre les différents algorithmes que nous avons implémentés, le calcul de plusieurs critères d'évaluation est effectué, notamment, **PSNR**, **SSIM**, ou **RMSE** (pour certain algorithmes) ainsi que le temps d'exécution et la qualité visuelle des images/vidéo. Il est à noter que l'implémentation de ces techniques de débruitage nécessite des outils et matériel software pour la programmation et le bon déroulement de l'exécution de ces derniers. Pour cela, nous nous sommes basés sur différents environnements et nous avons exploité le GPU à travers Google Colab surtout pour le cas de big data.

3.2. Architecture de réseau et implémentation de l'algorithme Auto-encodeur

Pour notre étude, nous avons réalisé un modèle d'architecture à auto-encodeur que nous avons introduit au 1er chapitre. Nous avons pu implémenter celui-ci grâce à la plateforme Google Colaboratory en raison des performances GPU qu'il procure, facilitant ainsi l'implémentation d'algorithme de ce genre. De ce fait le programme de cet algorithme est écrit en langage **Python** sous navigateur Anaconda. Avant le débruitage des données, nous avons effectué une étape de prétraitement pour ces données, comme nous le détaillerons dans ce qui suit.

3.2.1. Prétraitement des données (preprocessing)

Les données que nous avons utilisé dans notre étude de simulation, sont extraites de la base de données **MNIST** qui contient 70 000 exemples d'apprentissage avec une taille de 28×28. Comme il est important de bien traiter le dataset, nous avons divisé une partie de données pour l'entrainement (60 000 exemples) et une autre pour le test (10 000 exemples), le but étant de confirmer ainsi l'apprentissage par cet algorithme.

Nous avons aussi normalisé les valeurs contenues dans les images (qui sont des intensités de pixels) pour que le réseau puisse lire les données de façon standardisé entre 0 et 1.

3.2.2. Architecture Auto-encodeur utilisée

L'architecture adoptée pour notre algorithme est la suivante :

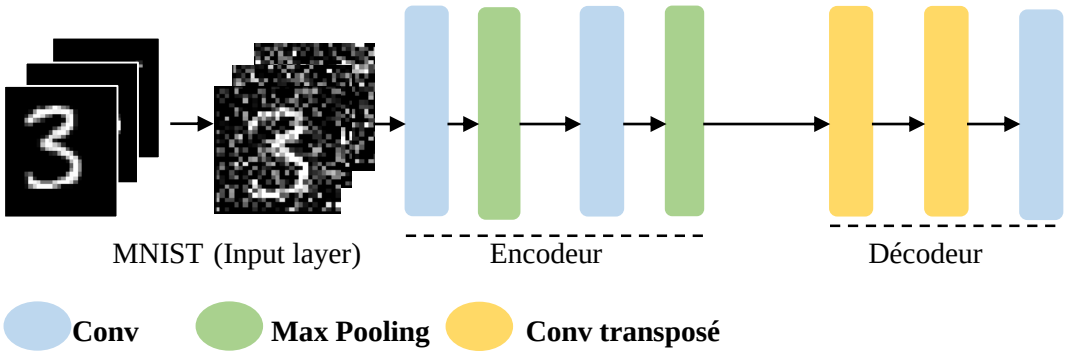


Figure 3. 1: Architecture adoptée pour l'implémentation de l'auto-encodeur.

Nos données qui sont les cellules d'entrées passent par une couche convolutive ensuite par une couche de sous-échantillonage par max pooling, ensuite une $2^{ème}$ fois par une couche convolutive suivie d'une deuxième couche de Pooling pour un deuxième sous échantillonnage. Celle-ci formant la partie d'encodage. Pour le décodage, il est formé de deux couches de convolution transposée, c'est-à-dire que nous avons fait une opération de sur-échantillonnage (Subsampling) pour permettre d'augmenter la résolution de l'image à la sortie qui passe par une dernière couche de convolution.

Nous avons conçu cet algorithme à partir de **Keras Tensorflow** qui possède une interface de programmation d'applications **API** (Application Pogramming Interface). Cette interface simplifie et facilite la conception de notre réseau de neurones qui peut être très complexe sans cette interface de programmation. Ce que contient notre réseau est le suivant :

- Des données d'entrée avec le bruit gaussien ajouté. Les images de taille $28 \times 28 \times 1$ puisque les images sont des binaires (noir et blanc) alors le canal vaut 1.
- La sortie de la première couche devient l'entrée de la couche suivante convolutive de l'encodeur, donc nous avons déterminé 32 filtres de **noyau** $\mathbf{3 \times 3}$, en ajoutant un **Padding** = 1 pour avoir en sortie de la couche des maps de même taille, la sortie de cette

couche après convolution donne 32 maps de taille 28×28 avec un canal de 32 maps ce qui donne $28 \times 28 \times 32$.

- Au passage vers la couche de Pooling un sous-échantillonnage de noyau de taille 2×2 est effectué avec un **Stride = 2** pour avoir en sortie (selon l'équation 1.12) une taille réduite qui vaut $14 \times 14 \times 32$.
- Un deuxième filtrage par la couche de convolution qui suit donne à sa sortie la même taille de $14 \times 14 \times 32$.
- Après un second sous-échantillonnage, de la même façon que le 1er, la taille devient encore plus réduite en sortie de cette couche $7 \times 7 \times 32$, par la suite elle devient l'entrée du décodeur.
- À l'entrée du décodeur, il y'a la couche de convolution transposée pour un sur-échantillonnage de taille $7 \times 7 \times 32$ vers une taille plus grande $14 \times 14 \times 32$ pour **stride = 2**.
- La deuxième couche convolutive transposée fait la même chose, c'est-à-dire de $14 \times 14 \times 32$ vers $28 \times 28 \times 32$ à sa sortie.
- Le filtrage dans la dernière couche permet de recouvrir à l'image initiale.

Nous avons donc résumé cette description par le schéma suivant :

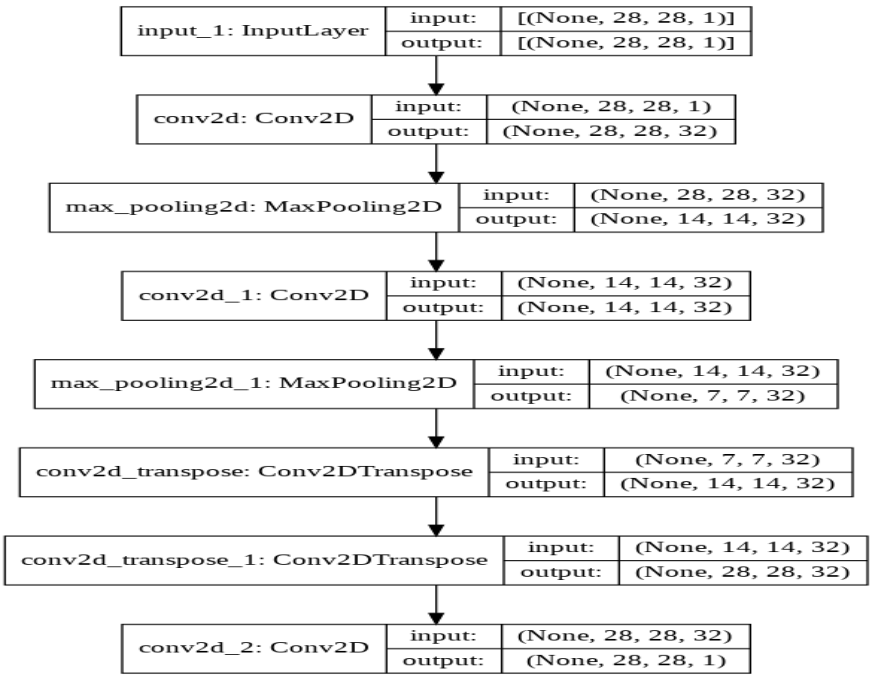


Figure 3.2: Schéma de la structure Auto-encodeur implémentée.

Nous ajoutons au final un nombre total de paramètres d'apprentissage à 28 353 (Poids synaptiques $\boldsymbol{W} = \mathbf{28\ 224}$ et biais $\boldsymbol{B} = \mathbf{129}$).

3.2.3. Résultats de l'implémentation

Les données d'apprentissage sont les suivants :

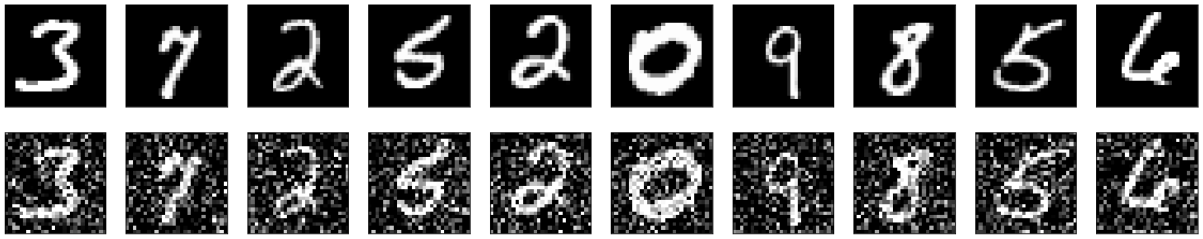
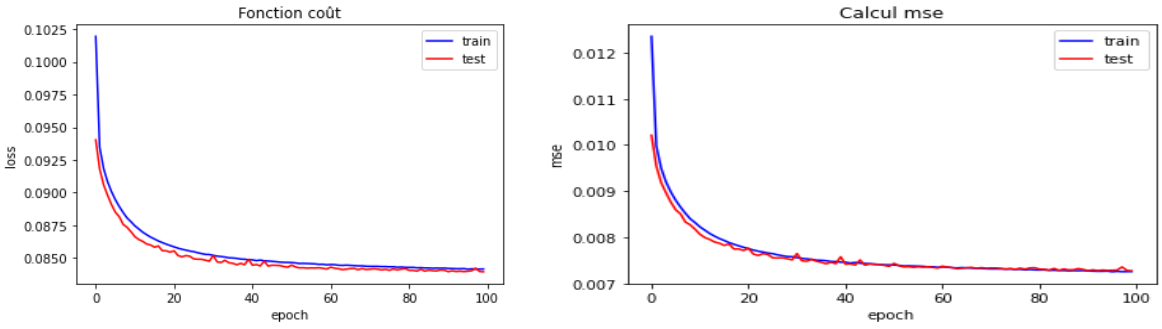


Figure 3.3: Images de Dataset importés MNIST.

Nous avons les images originales avec les images bruitées que nous avons généré, le bruit étant gaussien additif.

Nous avons traduit les résultats de débruitage au cours de l'apprentissage par des critères d'évaluation (metrics) selon les courbes de la fonction coût et le MSE obtenues qui sont les suivantes :



a) Développement de la fonction coût b) calcul de l'erreur quadratique moyenne

Figure 3.4: Résultats graphique de la fonction coût et du calcul MSE.

Au cours de l'apprentissage, suivant le développement du nombre d'itérations (epochs), c'est-à-dire que pendant les mises à jour des paramètres de notre modèle CNN, nous avons un développement de la fonction coût qui diminue au fur et à mesure que les itérations de forward et back propagation progressent, plus les itérations augmentent, plus la fonction coût est basse et converge vers 0. L'erreur quadratique moyenne diminue elle aussi, sachant que le calcul de MSE est la différence entre la valeur originale et la valeur prédite (predicted value) :

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(y - \hat{y})^2 \quad (3.\ 1)$$

Plus MSE diminue, meilleure est la précision **accuracy** (Figure 3.5) :

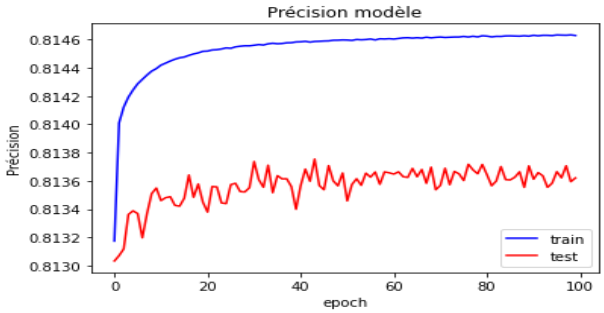


Figure 3. 5 : Précision du modèle (Accuracy).

Comme nous pouvons le remarquer sur cette figure, le modèle converge vers une précision de 81,5% à partir des données entraînées et ceux du test.

Nous avons ajouté de plus deux autres critères d'évaluation pour nos résultats, afin de vérifier la cohérence entre les images débruitées obtenues et celle reconstruite après sur-échantillonnage. Ces critères sont **le rapport crête signal sur bruit PSNR** (**P**eak **S**ignal **N**oise **R**atio) qui se mesure en **dB** en fonction de **MSE** afin voir la qualité et la fidélité de l'image obtenue par rapport à un signal bruité (Equation (3.2)). L'autre critère est l'index de similarité structurelle **SSIM** (**S**tructural **S**imilarity index) qui mesure le taux de similarité entre deux images numériques à partir de la comparaison structurelle, contraste et luminance. Plus les ces deux valeurs de PSNR et SSIM sont grandes mieux est l'algorithme.

$$PSNR = 10log_{10}(\frac{I_{max}^{\ 2}}{MSE}) \quad (3.\ 2)$$

$$SSIM(x, y) = l(x, y) \cdot c(x, y) \cdot s(x, y) \quad (3.\ 3)$$

Tableau 3. 1 : Résultat de débruitage en termes de PSNR et SSIM.

	PSNR	SSIM
Input	11.23 dB	53.47%
Output	21.45 dB	94.40%

Ce tableau montre la différence entre comment était la qualité de l'image avant passage par CNN où la qualité était détérioré jusqu'à ce qu'on lui applique le débruitage par auto-encodeur et de ce fait le taux de similitude arrive à 94.4% et un PSNR de 21.45dB.

Le résultat de prédiction des images bruitées en termes de qualité visuelle, est obtenu selon la correspondance entre chaque image :

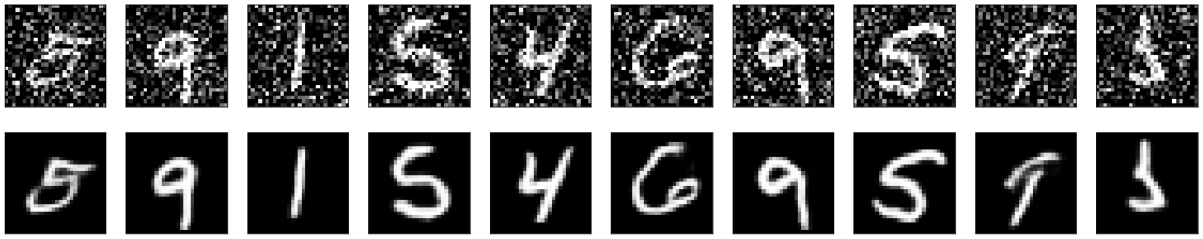


Figure 3.6: Résultat de la prédiction des données en termes de qualité visuelle.

Ce que nous venons de démontrer s'applique à des images, nous pouvons ramener des données de séquences vidéo pour des scènes et obtenir un débruitage adéquat en termes de filtrage et lissage. Pour le côté débruitage spatial des pixels, c'est identique au traitement des images, cependant, il y'aura un problème sur la cohérence temporelle de la trame T avec la trame $T + 1$ et $T - 1$. En raison de cela, nous allons implémenter les algorithmes de débruitage développés pour la vidéo et que nous avons introduit au chapitre 2. Nous allons vérifier qu'on obtienne une bonne qualité visuelle avec une bonne cohérence temporelle.

3.3. Données d'Apprentissage Utilisées

Les données ou Dataset à faire entrer dans nos algorithmes peuvent être des séquences d'images découpés et obtenues à partir de vidéos. Ces séquences peuvent être de plusieurs types d'images numériques et de formats variés (PNG, JPEG, Bitmap, JPG…).

Ces données seront bruitées avec un niveau de bruit additif blanc Gaussien de différentes valeurs de l'écart type **σ**. Nous analyserons l'évaluation du débruitage de ces séquences d'images selon chaque technique utilisé.

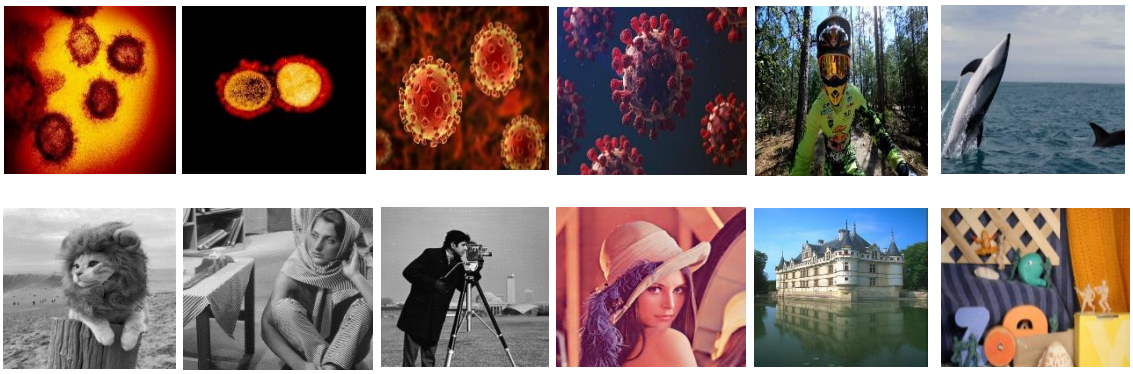


Figure 3.7: Données que nous avons utilisées.

Les deux dernières figures (en haut à droite) représentent une séquence d'image que nous avons utilisé pour le débruitage vidéo.

3.4. Algorithmes sélectionnés pour le débruitage image/vidéo

Nous considérons pour tous les algorithmes que le modèle adopté sur une image ou séquences d'images (Vidéos) est prise avec un bruit additif (Gaussien) tel que :

$$I = \hat{I} + N \tag{3. 4}$$

Où I une image bruitée observée, et $\hat{I}$ sa version originale que nous cherchons à récupérer (estimer). N un bruit blanc gaussien de moyenne nulle et d'écart type $\boldsymbol{\sigma}$.

Nous présentons dans ce qui suit les étapes d'implémentation des algorithmes d'images et vidéo que nous avons effectués au cours de ce travail.

3.5. Résultats et discussion

Nous présentons dans cette section la reproduction des résultats, ajouté à cela nos propres données personnelles d'images pour voir l'effet du débruitage des algorithmes sur celle-ci pour différentes valeurs de σ. L'étude de simulation est réalisée à travers un ordinateur avec **CPU intel core i5-4300U 1.90GHz 2.50 GHz 8Go RAM**, car la machine sur laquelle est implémenté les algorithmes influe directement sur le **Runtime**.

3.5.1. Débruitage images

3.5.1.1. BM3D

Nous avons utilisé l'algorithme BM3D sous **Matlab**. L'organigramme de ce dernier est illustré selon la figure suivante :

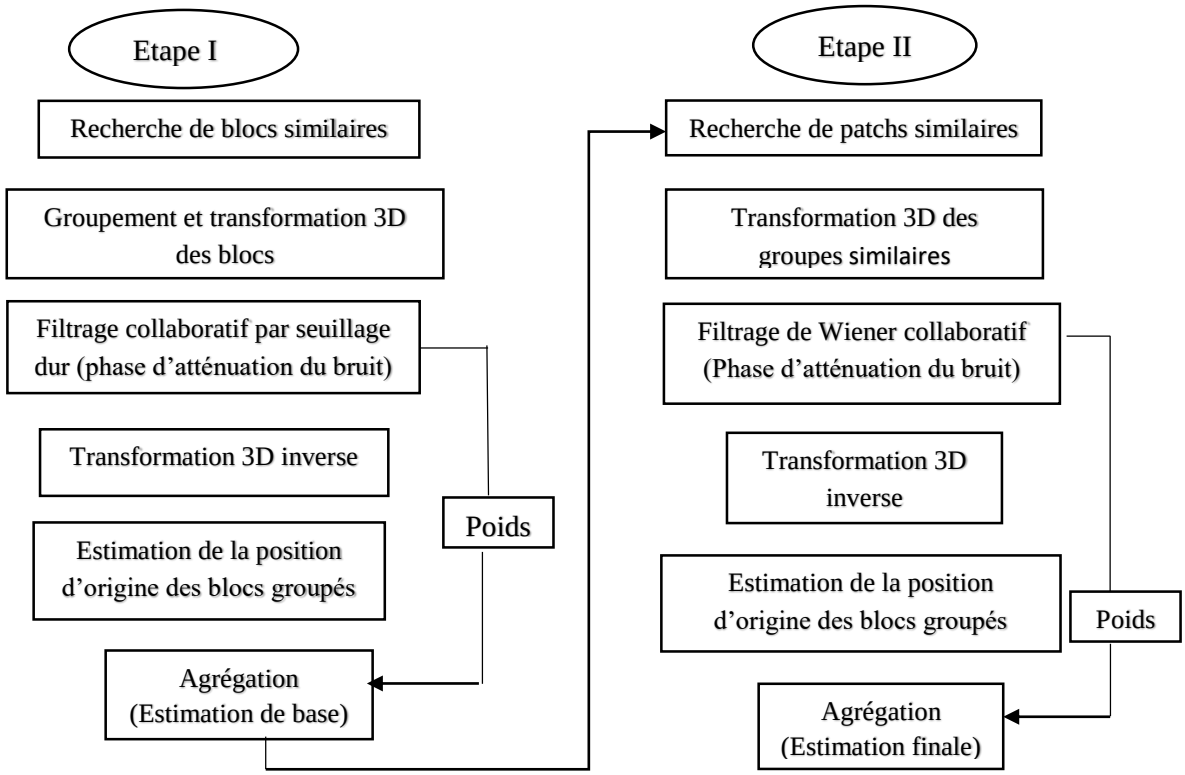


Figure 3. 8: Organigramme général de l'algorithme BM3D.

À partir des résultats sur des images en niveau de gris comme le montre celle appliquée sur l'image de « Lena » et celle du « cameraman » :

A- Cameraman (niveau de gris). B- Lena (Niveau de gris).

Figure 3. 9 : Expérience de débruitage d'image en niveau de gris de tailles 256x256 et sigma σ =40.

Nous remarquons un débruitage avec préservation des détails néanmoins, nous avons une légère différence par rapport aux images originaux, c'est pourquoi un calcul des estimations des correctifs ainsi que le taux de similitude entre l'image résultante et celle d'origine, est effectuée. À prendre en considération aussi que la taille de l'image (size) compte vis-à-vis du temps d'exécution du processus ainsi que la préservation des détails vu que dans notre algorithme, la distance entre les blocs similaires par rapport à un bloc de référence entre en facteur pour une estimation de débruitage. Le tableau suivant montre les différents résultats obtenus :

Tableau 3.2: Résultats du débruitage par BM3Dde l'image Cameraman et Lena en niveau de gris de taille 256x256.

	Cameraman			**Lena**		
Sigma σ	**PSNR (image bruitée à l'entrée)**	**PSNR (Estimation de base)**	**PSNR (Estimation finale)**	**PSNR (image bruitée à l'entrée)**	**PSNR (Estimation de base)**	**PSNR (Estimation finale)**
10	**28.1 dB**	**33.94 dB**	**34.18 dB**	**28.10 dB**	**33.74 dB**	**33.92 dB**
20	**22.08 dB**	**30.24 dB**	**30.48 dB**	**22.08 dB**	**30.09 dB**	**30.47 dB**
30	**18.56 dB**	**28.25 dB**	**28.64 dB**	**18.56 dB**	**28.14 dB**	**28.60 dB**
40	**16.06 dB**	**26.33 dB**	**27.18 dB**	**16.06 dB**	**26.25 dB**	**27.10 dB**
50	**14.12 dB**	**25.55 dB**	**26.12 dB**	**14.12 dB**	**25.64 dB**	**26.25 dB**

À partir de ce tableau nous avons 3 différents calcules du PSNR, Celui de l'entrée où le calcule est fait sur l'image bruitée par rapport à l'originale en fonction du niveau de l'écart type donné, et les deux PSNR (estimation de base et estimation finale) des deux sorties de chaque étape de l'architecture précédente. À noter aussi que c'est les résultats de l'estimation de base qui sont utilisés à l'entrée de la deuxième étape au lieu d'une image bruitée, ce qui permet de voir une différence sur les résultats. Les résultats montrent aussi que les valeurs du PSNR diminuent lorsque l'image est plus bruitée, on peut donc supposer que la reconstruction de l'image est plus détériorée et ne sera pas la même, c'est pourquoi nous vérifions l'effet du débruitage à travers le calcule du SSIM qui est illustré sur le tableau suivant :

Tableau 3.3: Résultats de temps d'exécution et similarité des images de Lena et Cameraman débruitées par BM3D.

	Cameraman		Lena	
Sigma σ	**SSIM**	**Temps d'exécution**	**SSIM**	**Temps d'exécution**
10	**93%**	1.1 s	**92%**	1 s
20	**87%**	1.1 s	**86%**	1.2 s
30	**83%**	1.5 s	**82%**	1.3 s
40	**80%**	1.3 s	**78%**	1.1 s
50	**77%**	1.8 s	**76%**	1.5 s

Nous remarquons que le taux de correspondance de l'image diminue d'une similarité de 92% pour un écart type de 10 à une similarité de 76% pour un écart type σ de 50, ce qui est relativement compréhensible, étant donné que le taux de bruit augmente. Nous pouvons aussi expliquer un autre fait qui est celui de l'estimation de la position retournée lors de la transformation 3D inverse sur lequel la superposition des blocs filtrés de l'image lors du seuillage dur ou filtrage de Wiener induit une estimation de chaque pixel et différentes variances, et par conséquent, l'usage d'un calcule d'agrégation par poids moyen (comme sur la figure du schéma) de toute l'image. C'est de là qu'on observe une agrégation dans l'étape 1 et à partir de celle-ci on l'utilise pour piloter la 2ème partie de débruitage, ce qui permet d'avoir une meilleure estimation des blocs avec un meilleur PSNR dans la 2ème partie et donc une meilleure qualité de l'image débruitée.

L'algorithme a été testé de même sur une image RGB de taille identique que les deux précédentes images en niveau de gris afin de voir l'effet sur l'exécution et la qualité qui en résulte, la figure qui suit montre le résultat de Lena en RGB 256x256.

Figure 3.10 : Expérience de débruitage d'image Lena en couleur RGB pour σ = 40.

Tableau 3.4: Résultats du débruitage de l'image Lena en RGB 256x256.

Sigma σ	PSNR (image bruitée à l'entrée)	PSNR (Estimation de base)	PSNR (Estimation finale)	Temps d'exécution	SSIM
10	**28.14 dB**	**33.93 dB**	**33.99 dB**	2.1 s	**90%**
20	**22.11dB**	**30.92 dB**	**31.17 dB**	1.7 s	**86%**
30	**18.6 dB**	**29.07 dB**	**29.44 dB**	1.9 s	**82%**
40	**16.09 dB**	**27.77 dB**	**27.91 dB**	1.1 s	**78%**
50	**14.16 dB**	**26.84 dB**	**27.38 dB**	3.1 s	**77%**

À partir du résultat obtenu de l'image Lena RGB 256x256 on remarque une atténuation du bruit et la préservation des détails et des caractéristiques de l'image. De même, les informations illustrées sur le tableau montrent des résultats avec une légère différence au niveau du SSIM que ceux obtenus pour les images en niveau de gris. Néanmoins, nous pouvons visualiser le chapeau préservé avec les quelques régions lisses de l'image. À noter que le bruit dans le cas de cette extension sur l'algorithme pour les images RGB que le bruit appliqué ici sera sur plusieurs canaux séparés qui sont le signal de chrominance U et V et de luminance Y, ainsi, la technique appliquée pour ce cas est faite séparément car les propriétés de chaque canal se diffèrent sur le signal informatif et la fréquence appliquée, ce qui peut influer sur l'estimation et du PSNR, cependant, le principe reste le même.

Nous avons observé des tests sur des données extérieurs que nous avons proposé et de voir de la même manière les résultats obtenus.

L'image suivante présente une image qui contient plusieurs régions assez grisâtres et un fond d'image avec moins de parties sombres et noires que nous appellerons «Simba» :

Figure 3.11 : Débruitage d'image en niveau de gris par BM3D, niveau du bruit σ = 50.

Le résultat de ce débruitage montre un lissage assez conséquent qui peut être dû en raison des détails qui sont quasiment très similaires au niveau des recoins de chaque région des détails, donc les patchs similaires lors de l'estimation et de l'agrégation peuvent mener à un résulter où on voit une image débruitée avec lissage dans l'image qui est assez visible.

Tableau 3. 5: Résultats du débruitage de l'image Simba par BM3D.

Sigma σ	PSNR (image bruitée à l'entrée)	PSNR (Estimation de base)	PSNR (Estimation finale)	Temps d'exécution	SSIM
10	**28.10 dB**	**33.09 dB**	**33.41 dB**	1 s	**92%**
20	**22.08 dB**	**29.67 dB**	**30.03 dB**	1.1 s	**85%**
30	**18.56 dB**	**27.82 dB**	**28.29 dB**	1.3 s	**79%**
40	**16.06 dB**	**26.36 dB**	**27.04 dB**	1 s	**73%**
50	**14.12 dB**	**25.59 dB**	**26.24 dB**	1.9 s	**70%**

En comparaison avec le tableau précédent sur les deux images en niveau de gris, on remarque que les résultats du SSIM commencent à s'écarter par rapport à ceux du tableau précédent dès que σ vaut 30 et au-delà, on a une similarité qui diminue de 30% par exemple lorsque σ=50.

D'autres données utilisées en RGB pour des images du Covid, où on a importé des tailles assez grandes par rapport à celle précédemment utilisées. L'image de gauche de taille 640x360 et celle de droite de taille 970×647. À partir de là on observe les résultats des tableaux en terme de débruitage, filtrage et préservation des détails par rapport à l'originale.

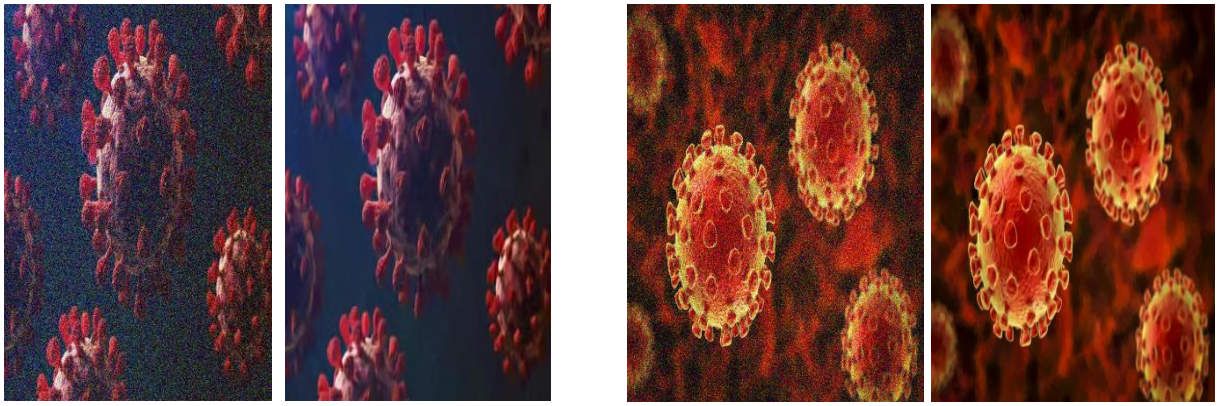


Figure 3.12 : Débruitage de données personnelles d'images en couleurs (Exemple de Covid-19) pour σ = 50 par BM3D.

Nous avons un résultat qui montre une bonne préservation des détails et de la qualité d'image, cela correspond parfaitement aux résultats des tableaux suivant :

Tableau 3. 6 : Résultats des tests de débruitage des deux images du Covid-19 par BM3D.

	Covid (Image de gauche)			**Covid (Image de droite)**		
Sigma *σ*	**PSNR (image bruitée à l'entrée)**	**PSNR (Estimation de base)**	**PSNR (Estimation finale)**	**PSNR (image bruitée à l'entrée)**	**PSNR (Estimation de base)**	**PSNR (Estimation finale)**
10	**28.14 dB**	**37.03 dB**	**37.58 dB**	**28.14 dB**	**39.03 dB**	**39.70 dB**
20	**22.12 dB**	**33.02 dB**	**33.73 dB**	**22.12 dB**	**34.98 dB**	**35.97 dB**
30	**18.6 dB**	**30.76 dB**	**31.57 dB**	**18.59 dB**	**32.59 dB**	**33.74 dB**
40	**16.1 dB**	**29.21 dB**	**29.95 dB**	**16.1 dB**	**30.88 dB**	**31.90 dB**
50	**14.16 dB**	**28.11 dB**	**29.16 dB**	**16.16 dB**	**29.82 dB**	**31.33 dB**

Dans ce tableau nous avons ici un PSNR plus satisfaisant lorsqu'on compare celui de l'image bruitée à l'entrée, avec ceux de la sortie où nous apercevons à chaque croissance du niveau de bruit l'écarte est plus au moins étendu, en plus de l'estimation finale qui permet d'améliorer encore plus notamment lorsque le niveau de l'écart type est grand on peut atteindre une différence de 1dB (de 28.11dB à 29.16dB). De même pour l'image du covid de droite où nous avons plus de correctifs étant donné que nous avons des valeurs du PSNR plus croissants ajouté à cela, l'estimation finale à travers le filtrage de Wiener empirique en supposant que l'atténuation via l'estimation de base comme par exemple pour σ = 50, nous remarquons une amélioration de 2dB environ entre les deux étapes de l'algorithme. Tout cela en notant de plus

que l'image de droite possède une taille plus grande que celle de gauche, et donc nous supposons que le nombre de patchs similaires augmente ainsi que l'estimation de la moyenne de l'agrégation finale avec.

Tableau 3.7: Résultats du débruitage de l'image Covid-19 par BM3D.

	Covid-19 (Image de gauche)		Covid-19 (Image de droite)	
Sigma σ	**SSIM**	**Temps d'exécution**	**SSIM**	**Temps d'exécution**
10	**94%**	**7.5 s**	**96%**	**20 s**
20	**90%**	**7.6 s**	**93%**	**21.3 s**
30	**86%**	**7.5 s**	**90%**	**21 s**
40	**82%**	**5.5 s**	**85%**	**15 s**
50	**81%**	**11.4 s**	**86%**	**34.9 s**

Vu que nous avons un débruitage satisfaisant, il est accompagné d'un taux de similitude très proche de l'image originale, en remarquant aussi que l'image de droite possède un taux plus grand que celui de gauche, il reste néanmoins que le taux de similarité SSIM plus grand que l'image de gauche allant jusqu'à une différence de 5% entre les deux lorsque l'écart type σ augmente.

Le niveau du bruit peut être augmenté jusqu'à un écart de σ =100.

3.5.1.2. DnCNN

Nous pouvons voir et de façon efficace à partir de la figure qui suit, l'architecture de l'algorithme DnCNN :

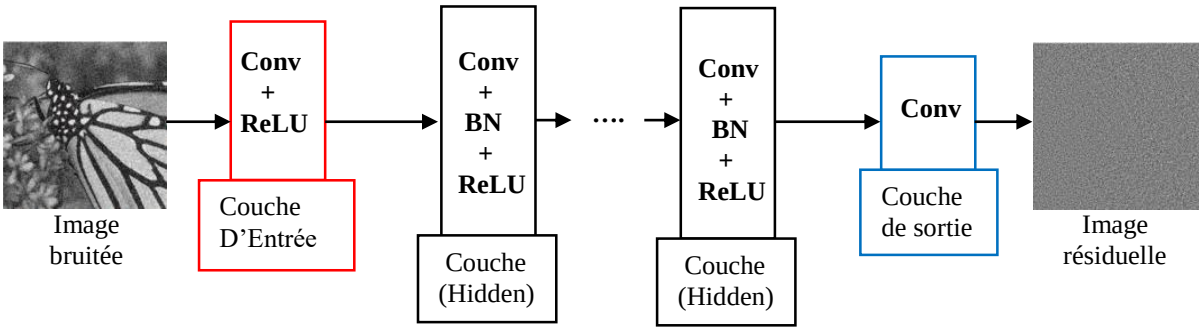


Figure 3. 13 : Architecture du réseau DnCNN.

L'architecture comprend trois types de couches : la première couche d'entrée (encadrée en rouge) contient une fonction de convolution (Conv) avec 64 filtres (ou Kernels) de taille ($3 \times 3 \times C$) pour générer 64 cartes de caractéristiques de l'image, et une fonction d'activation de non-linéarité appelée Unité Linéaire Rectifiée (**ReLU** pour **Rectified Linear Unit**) défini par $f(x) = max(0, x)$ pour tout réel x. $\boldsymbol{C}$ représente le niveau d'image (gris ou couleur) de l'image si c'est une image en niveau de gris dans ce cas $\boldsymbol{C = 1}$ et pour une image en couleur $\boldsymbol{C = 3}$. Le réseau est défini par une profondeur (depth) de D = 17 c'est-à-dire une dimension de $2D+1 \times 2D+1$ (35×35) donc, en allant de la 2ème couche à ($D-1$), on trouve des couches cachées (Hidden layers) qui contiennent la fonction de convolution et la fonction d'activation, la normalisation par batch tout au long des couches hidden layers avec 64 filtres de taille $3 \times 3 \times 64$. Dans la dernière couche (encadrée en bleu), on trouve une couche de convolution où on a $\boldsymbol{C}$ filtres (en fonction de $C = 1$ ou $C = 3$) de taille $3 \times 3 \times 64$ pour la reconstruction de la sortie en image résiduelle selon l'apprentissage Residual par la soustraction entre l'image bruitée et cette dernière couche du modèle. Il est important aussi de remarquer à travers la Figure2.6, que la couche de Pooling est enlevée de cette architecture DnCNN.

Nous avons utilisé l'algorithme DnCNN avec Python sous trois valeurs de l'écart type, sachant que cet algorithme possède un écart type qui peut être régler dans l'intervalle $\sigma \in [0,55]$.

L'algorithme DnCNN ayant été entraîné sur 400 images de taille 180×180 avec une taille de batch de 128 et le nombre d'epochs (itérations) 50. Le nombre total de patches à entrainer est de $128 \times 3000 = 384\,000$ patches.

L'étape d'apprentissage des données est résumée comme suit :

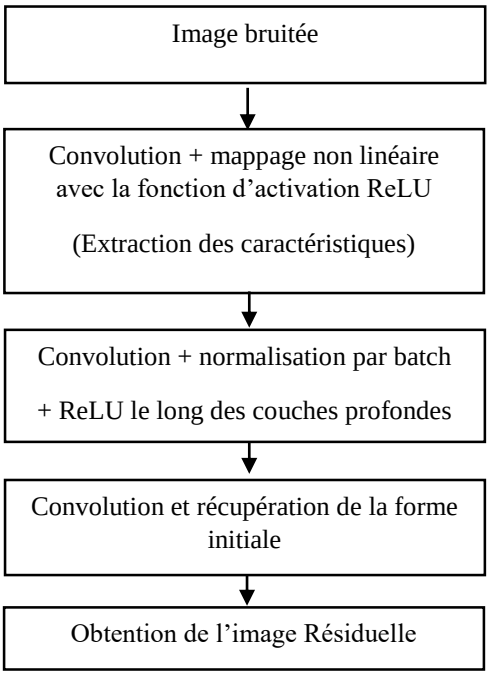


Figure 3.14: Organigramme du réseau de neurones DnCNN.

Les données utilisées sur [24] testés à l'aide de la fenêtre d'invite de commande Prompte que procure **Anaconda 3** où on exécute le programme en spécifiant le répertoire des données en question et du fichier python, comme par exemple :

" Python main_test.py – set_dir Data/Test -- sigma 15 – save_result "

"Python main_train.py -- batch_size 128 – train_data Data/train400 –epoch 300 – lr 1e-3"

Nous avons donc dans le premier exemple défini un test sur une image en lui ajoutant un bruit blanc gaussien d'écart type qui vaut 15 en sauvegardant les résultats du débruitage dans un répertoire par défaut. Dans le 2ème exemple, nous avons initialisé un learning rate (taux d'apprentissage) à 1×10^{-3} et spécifié le répertoire du dataset à apprendre pour notre structure, ainsi que le taille batch_size pour le nombre de données à traîter pendant les itérations d'apprentissage par epoch = 300. À noter que cette structure est construite par la bibliothèque **Pytorch** et non la bibliothèque **Tensorflow**.

Les résultats sont illustrés suivant le résultat de l'image et du tableau suivant :

Figure 3. 15: Images utilisés pour tester l'algorithme DnCNN avec σ = 25.

Nous remarquons que le filtrage de l'image a bien été effectué, il reste alors à voir en terme de qualité de l'image à travers le calcul des critères d'évaluation qui sont traduit dans le tableau suivant pour chaque image (Lena, Cameraman, Papillon, Peroquet, et Maison) :

Tableau 3. 8: Résultats des critères d'évaluation des différentes images utilisés sur DnCNN.

	Cameraman		**Maison**		**Papillon**		**Perroquet**		**Lena**	
Ecart type σ	**PSNR (dB)**	**SSIM (%)**	**PSNR (dB)**	**SSIM (%)**	**PSNR (dB)**	**SSIM (%)**	**PSNR (dB)**	**SSIM (%)**	**PSNR (dB)**	**SSIM (%)**
15	**30.09**	**91.31**	**33.5**	**94.26**	**30.56**	**95.24**	**29.39**	**91.30**	**32.32**	**93.55**
25	**30.26**	**93.17**	**33.14**	**94.39**	**30.42**	**95.57**	**29.45**	**92.66**	**32.4**	**94.16**

La qualité d'image semble bonne étant donné que nous avons un SSIM supérieur à 90% sur l'ensemble des images et pour deux valeurs différentes de l'écart type.

Pour ce qui est de nos propres données :

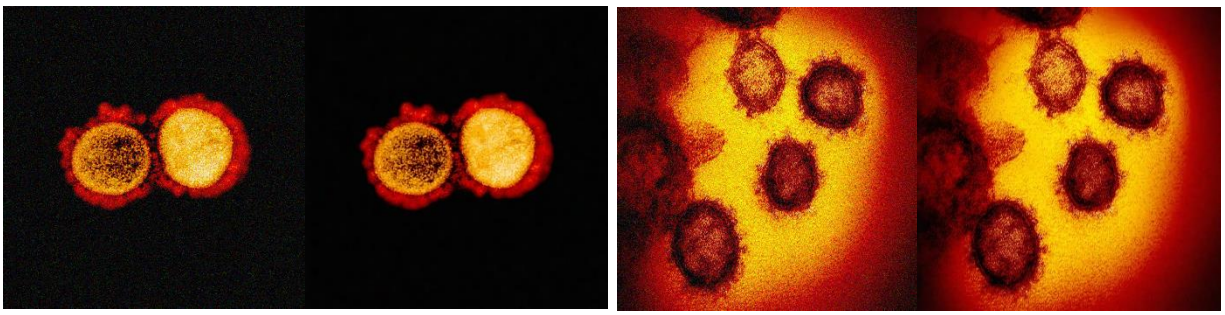


Figure 3. 16: Image bruitée et débruitée par DnCNN de taille 1024x1024 (σ = 30).

lorsque la taille de l'image devient grande, la qualité et le PSNR est plus grand en raison du bruit dans les pixels qui est camouflé face aux détails de l'image en prenant aussi en considération que lorsque l'image est plus sombre, le débruitage devient plus simple lorsque nous remarquons les résultats PSNR de l'image sombre de gauche sachant que les deux sont de même taille :

Tableau 3. 9 : Résultats de nos propres données importées utilisés sur DnCNN.

	Sras-Cov-2 (Sombre)			Sras-Cov-2 (Clair)		
Ecart type σ	**PSNR (dB)**	**SSIM (%)**	**Temps d'exécution (s)**	**PSNR (dB)**	**SSIM (%)**	**Temps d'exécution (s)**
15	**35.77**	**76.94**	**38.84**	**29.25**	**81.18**	**48.9**
25	**35.43**	**76.42**	**42.16**	**29.59**	**83.36**	**41.47**
30	**32.6**	**73.58**	**42.33**	**27.42**	**77.50**	**40.7**

Il faut savoir qu'en principe, le code de cet algorithme marche sous une machine avec GPU, c'est pourquoi, nous avons modifié une partie du code pour qu'il puisse se lancer en mode CPU sur une machine qui n'intègre pas de GPU, de manière générale, nous avons changé le package CUDA (qui est un package dédié par NVIDIA pour utiliser les ressources d'une carte graphique GPU).

3.5.1.3. FFDNet

L'architecture FFDnet partage la même structure que DnCNN pour les couches convolitifs comme on peut le voir sur la figure suivante incluant avec elle le prétraitement et le post traitement de la résolution de l'image :

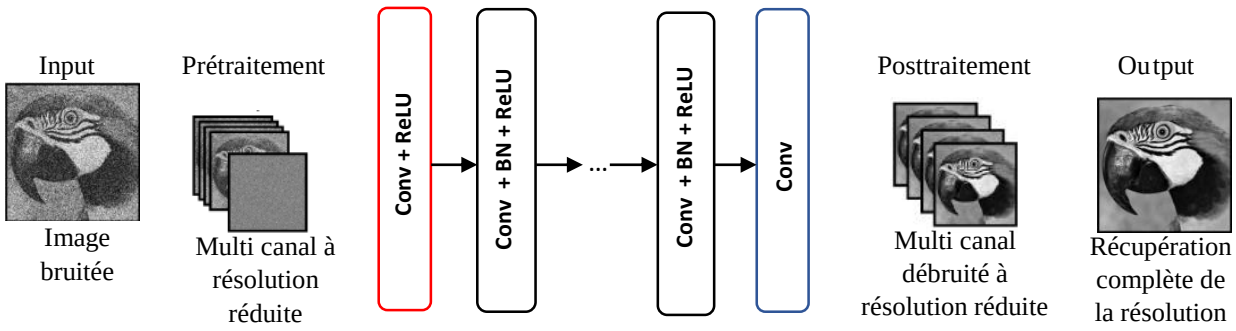


Figure 3. 17: Architecture de FFDnet.

Contrairement au DnCNN, l'algorithme FFDnet a été entraîné plus, avec un total de 1 024 000 patches de taille 64×64(Pour un batch de taille 128 par 8000) sur ce qui bien sur veut dire que le nombre d'itérations augmente. La taille du noyau de filtre utilisé est 3×3. Nous pouvons varier l'écart type pour le bruit entre σ du bruit entre 0 et 75.

Le processus de débruitage passe par les mêmes étapes lors de l'apprentissage excepté que celui-ci rajoute 2 phases comme cités dans le chapitre 2.

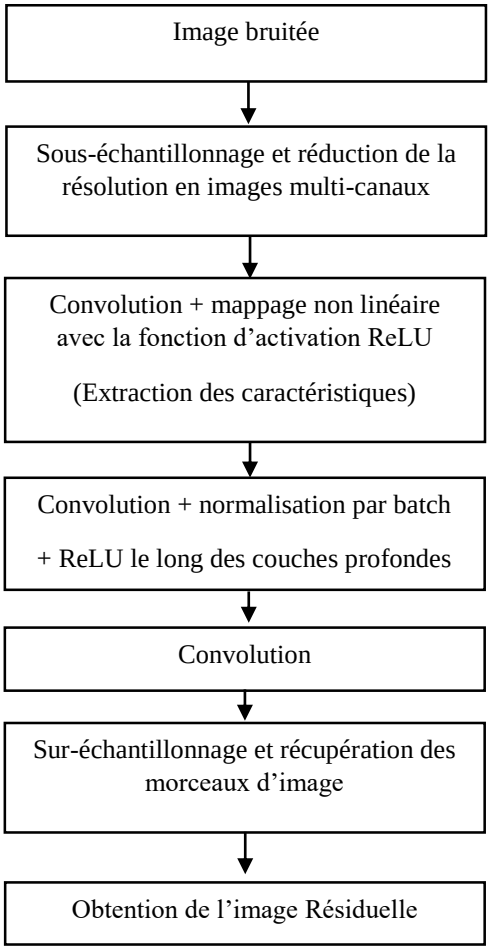


Figure 3.18: Organigramme de débruitage FFDnet.

La même chose ici pour FFDNet qui marche de la même manière (conçu avec Pytorch). Nous avons utilisé l'image de Barbara (niveau de gris) et House (RGB) dans [26].

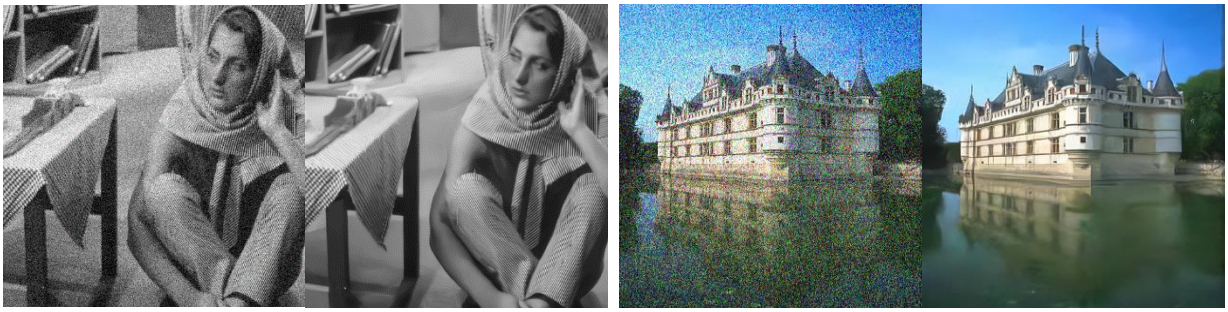


Figure 3. 19 : Débruitage en niveau de gris σ = 25 et débruitage par FFDnet image RGB σ = 55.

Nous avons une bonne préservation des détails et des bords de l'image, nous avons aussi une partie lissante dans l'image en couleur après filtrage. Selon le tableau suivant, nous avons une bonne similitude entre l'image originale et l'image résultante ainsi que les résultats qui se ressemblent avec [26].

Tableau 3. 10 : Résultats des critères d'évaluation pour débruitage Lena et Maison avec FFDnet.

	Barbara (Niveau de gris)		**Maison (RGB)**	
Ecart type σ	**PSNR (dB)**	**SSIM (%)**	**PSNR (dB)**	**SSIM (%)**
15	**30.09**	**91.32**	**30.16**	**91.95**
25	**30.11**	**88.5**	**30.12**	**93.06**
30	**26.73**	**81.34**	**26.79**	**80.14**

Sur nos données, nous avons utilisé l'écart type le plus élevé par FFDnet, c'est-à-dire σ=75 :

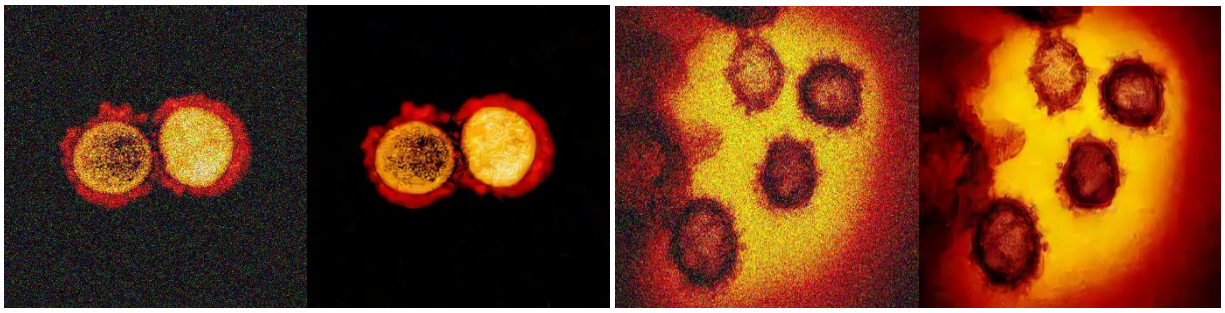


Figure 3. 20 : Débruitage par FFDnet des images du Sras-Cov-2, sombre à gauche et claire à droite (512x512 σ=75).

Nous pouvons bien voir le bruit appliqué et voir la différence entre le débruitage sur une image bien claire par rapport à une autre qui possède des pixels assombris :

Tableau 3. 11: Résultats d'évaluation des images Sras-Cov-2 par FFDnet.

	Sras-Cov-2 (Sombre)			Sras-Cov-2 (Clair)		
Ecart type σ	**PSNR (dB)**	**SSIM (%)**	**Temps d'exécution (s)**	**PSNR (dB)**	**SSIM (%)**	**Temps d'exécution (s)**
15	**37.32**	**97.03**	**5.11**	**31.39**	**84.4**	**5.24**
25	**34.77**	**95.29**	**5.2**	**29**	**76.58**	**5.25**
30	**33.95**	**94.31**	**5.86**	**28.20**	**73.18**	**5.1**
55	**31.16**	**88.98**	**5.08**	**26.01**	**61.88**	**5.55**
75	**29.94**	**83.75**	**5.08**	**25.11**	**56.26**	**5.32**

Le débruitage se fait plus clairement pour l'image avers les bords de régions préservés par rapport à celle de droite.

3.5.2. Résultats du débruitage Vidéo

Les Vidéos utilisés comme données pour le débruitage vidéo se doivent d'être traités et découpés en séquences d'images grâce à un module ou logiciel spécifique nommé : **FFMPEG** que nous avons utilisé.

Celui-ci permet d'appliquer un traitement audio et vidéo, pour notre cas, la lecture d'une séquence vidéo en entrée dans le module suivant des options de commandes désirées avec un format et codec spécifique pour avoir en sortie un nouveau type de données qui est plusieurs séquences d'images découpées avec un format défini, ou un nouveau format de vidéos :

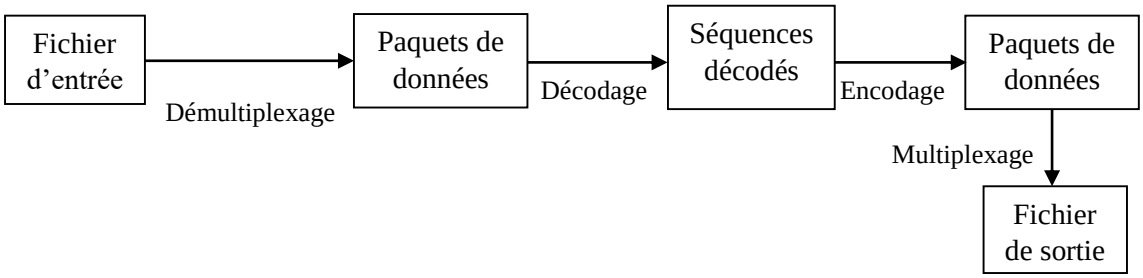


Figure 3.21: Schéma descriptif du procédé de conversion par FFMPEG des données vidéo.

Grâce à la librairie **libavformat** dont fait appel le logiciel contenant un démultiplexeur pour lire le fichier et obtenir les paquets de données encodés qui passeront par un décodage produisant des séquences non compressés. Ensuite avec une technique de filtrage, les séquences

sont encodées en paquets de données pour passer vers la sortie d'une nouvelle forme de fichier via un multiplexage.

La figure suivante présente la fenêtre d'utilisation du logiciel FFMPEG (à partir du Prompt d'Anaconda) :

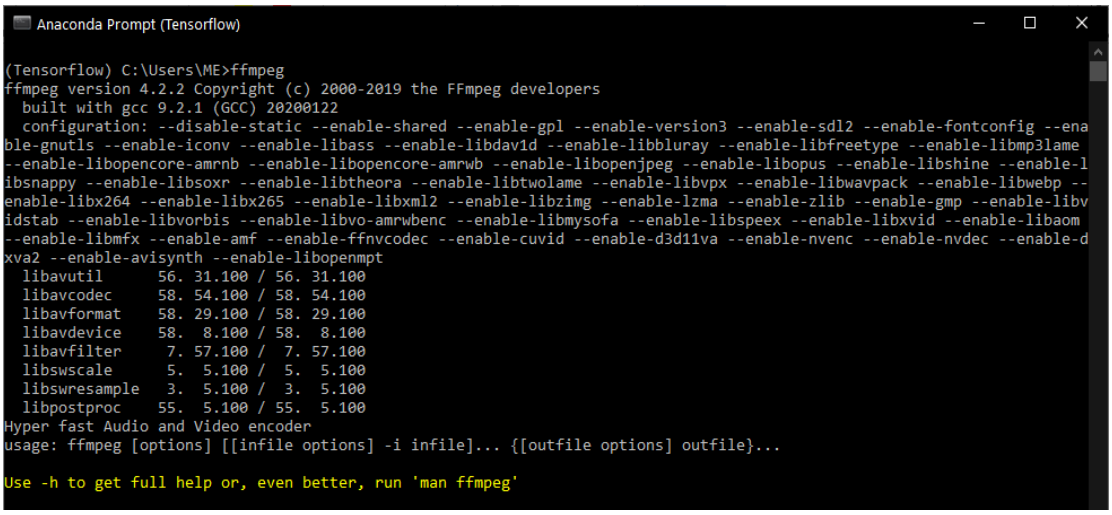


Figure 3. 22 : Invite de commande Anaconda pour l'utilisation de FFMPEG.

La syntaxe générale pour utiliser FFMPEG est la suivante :

ffmpeg [[infile options][**-i** *infile*]]... {[outfile options] *outfile*}...

Exemple:

ffmpeg -i input.avi -r 1 -s WxH -f image2 output-%03d.jpeg

Où l'argument -i désigne l'entrée, suivi d'options (facultatifs) comme –r désigne le ratio de découpage c'est-à-dire la fréquence de découpage de la vidéo en séquences d'images (en Hz). –s désigne la taille des images en sortie qui est par défaut la même que la source. Enfin –f force le format en sortie (.jpeg).

Il faut savoir qu'il existe plusieurs abréviation en tant qu'options introduit comme argument pour appliquer le traitement de conversion sur les données avec FFMPEG, il est donc conseiller de consulter la documentation qui est disponible sur le site officiel de la distribution du logiciel.

Le découpage en séquences d'image peut être selon une fréquence déterminée, plus la fréquence est haute plus le nombre de séquences augmente et le mouvement ou le flux dans la vidéo paraît plus large.

3.5.2.1. SPTWO

Pour l'algorithme SPTWO, nous l'avons exécuté sous Linux, via une machine virtuelle **Oracle VM VirtualBox,** La version minimum pour pouvoir exécuter SPTWO avec tous les packages qu'il faut est la version 16.04 de Ubuntu.

L'algorithme se base sur le débruitage par patch comme suit :

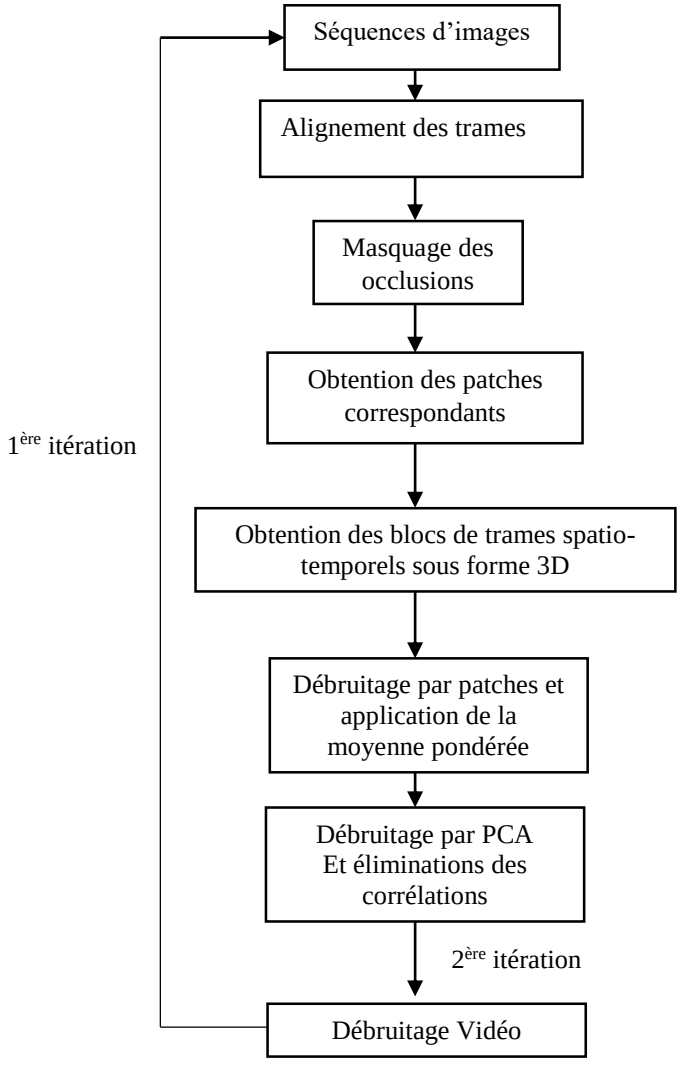


Figure 3. 23 : Organigramme de débruitage SPTWO.

En principe, cet algorithme calcul le critère d'évaluation **RMSE (R**oot **M**ean **S**quare **E**rror**) :**

$$RMSE = \sqrt{\sum_{i=1}^{n} \frac{(\hat{y}_i - y_i)^2}{n}} \quad (3.5)$$

Cependant, nous avons rajouté le calcul du PSNR afin de mieux comparé par la suite avec les autres algorithmes de débruitage Vidéo. Comme nous l'avons mentionné plus haut, cet algorithme a été testé sous Linux, nous avons donc compilé le programme depuis l'invite de commande dans le répertoire du programme en question que nous avons compilé par la directive OMP=1 qui crée les fichiers exécutables appropriés pour la culcul RMSE, le débruitage SPTWO et l'ajout du bruit Gaussien. Nous avons mis une séquence d'images dans un répertoire à définir dans un dossier à partir duquel le programme puisse lire la séquence en question. La figure suivante montre une séquence originale que nous avons utilisée comme sur [37].

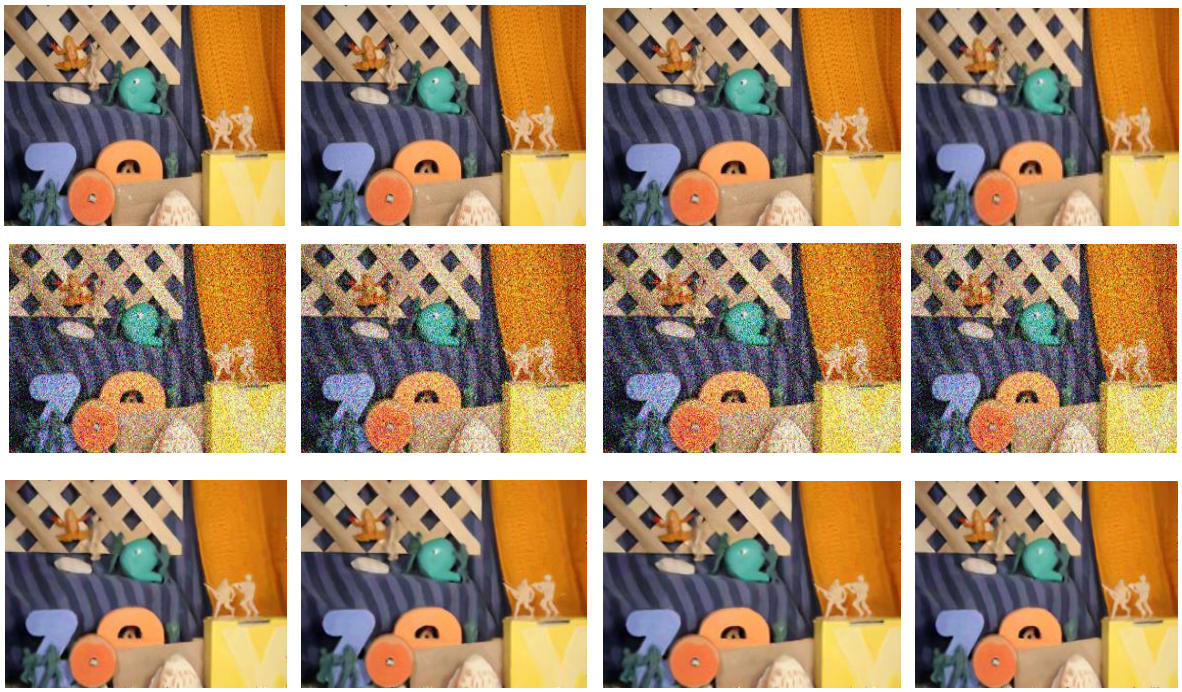


Figure 3. 24 : Débruitage de la séquence appelée « Army » avec σ = 50 par SPTWO.

Nous avons donc un débruitage pour toutes les trames correspondantes. De haut en bas nous avons la séquence originale suivie de la séquence bruitée et en bas le résultat obtenu. Les résultats de différentes valeurs de σ montrent pour le cas du **RMSE,** que plus il est petit mieux est le débruitage.

Tableau 3. 12: Résultats d'évaluation des images Army par SPTWO.

Ecart type σ	RMSE	SSIM (%)	PSNR (dB)
10	**3.59**	**95.60**	**37.03**
20	**5.204**	**91.56**	**33.75**
30	**6.36**	**88.61**	**32.01**
40	**7.26**	**86.49**	**30.84**
50	**8.16**	**84.1**	**29.72**

3.5.2.2. VNLB

Nous avons également implémenté le VNLB sous Linux Ubuntu 16.04, nécessitant aussi des packages spécifiques dédiés à l'algèbre linéaire pour les calculs dans le programme. Les packages sont LAPACKE et CBLAS.

Comme pour le cas de l'algorithme BM3D, deux étapes principales d'estimation se présentent :

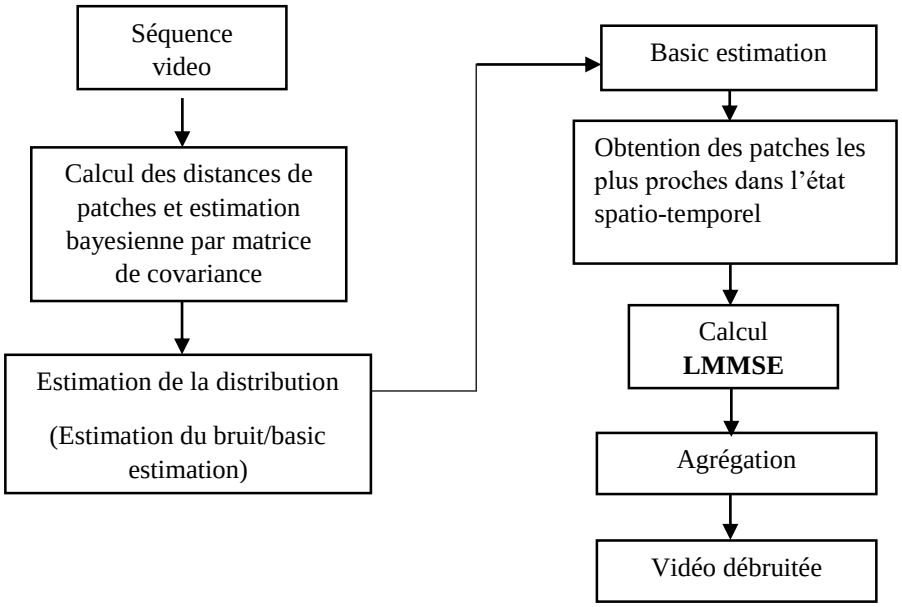


Figure 3.25 : Organigramme de débruitage VNLB.

De la même manière pour l'algorithme VNLB, où nous l'avons utilisé sous Linux Ubuntu, en installant les packages nécessaires comme LAPACK et CBLAS. Nous avons utilisé la même séquence c'est-à-dire la séquence « Army » dans un soucis de comparaison.

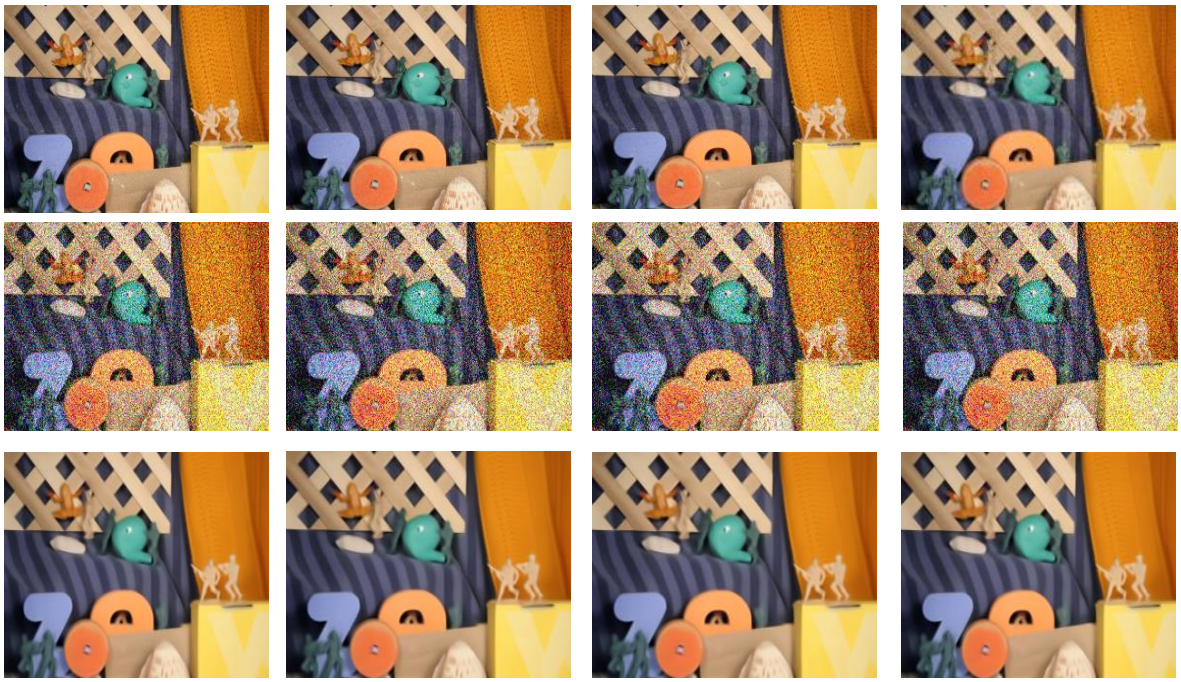


Figure 3. 26 : Débruitage de la séquence appelée « Army » avec σ = 50 par VNLB.

De même, le calcul des critères d'évaluation est le suivant :

Tableau 3. 13 : Résultats d'évaluation des images Army par VNLB.

Ecart type σ	PSNR (dB)	SSIM (%)
10	**37.51**	**95.24**
20	**34.45**	**91.51**
30	**32.93**	**89.15**
40	**32.09**	**88.3**
50	**31.2**	**87.13**

3.5.2.3. VBM4D

Nous avons utilisé cet algorithme à l'aide de Matlab, nous pouvons directement introduire une vidéo Brute sans découpage en plusieurs séquences d'images, puisqu'il existe des toolbox de traitement vidéo sur Matlab. Le principe de l'algorithme hérité à partir du BM3D reste le même, c'est-à-dire qu'après le **groupement**, le **filtrage collaboratif** entre en jeu en transformant chaque groupe par une dé-corrélation séparable $4D$ puis par **seuillage** et **transformation inverse** selon l'ordre des étapes de la figure 3.8. De cette façon les groupements peuvent donc par **agrégation** retourner à leur position initiale dans la vidéo.

Nous avons extrait une photo du débruitage qui est en niveaux de gris sur la figure qui suit, car le programme traite les séquences en les convertissant en niveaux de gris :

Figure 3. 27 : Débruitage par VBM4D σ = 50.

Tableau 3.14: Résultats d'évaluation par VBM4D.

Ecart type σ	PSNR (dB)	SSIM (%)	RUN (s)
10	**41.31**	**95.15**	**409**
20	**36.76**	**92.57**	**463.4**
30	**34.33**	**89.68**	**434**
40	**32.56**	**86.9**	**425.7**
50	**31.77**	**84.05**	**451.5**

Nous avons ici un débruitage de bonne qualité puisque nous avons un PSNR arrivant à 41.31dB sur un écart type de 10 et le SSIM touchant un taux de 95.15%. La qualité et la préservation des détails restent aussi pour le cas de σ = 50.

3.5.2.4. DVDNet

Pour DVDnet, nous l'avons implémenté en langage Python, 1 024 000 données d'entraînements (comme sur FFDnet), la taille des patchs vaut 50×50 qui vont être contaminées par un bruit blanc gaussien $\sigma \in [0,55]$. Le réseau de neurone CNN du DVDnet va traîter les opérations de l'organigramme suivant :

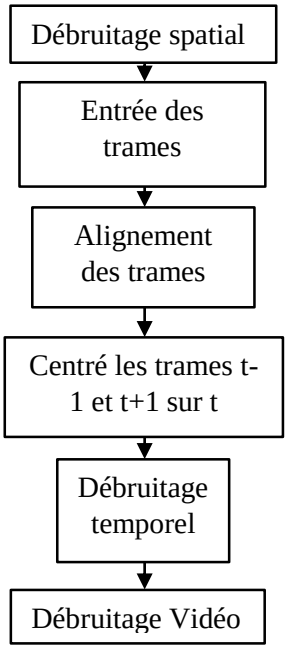


Figure 3. 28 : Organigramme de débruitage DVDnet.

Concernant les détails de cette architecture, il faut savoir que le bruit vient avec les trames d'entrée. Les blocs de débruitage spatial sont composés de **12 couches convolutionnelles**, et **6 couches convolutionnelles** pour le débruitage temporel (Figure 3.29). Cette architecture est prise de l'algorithme FFDnet de base, puisqu'elle comprend aussi un sous échantillonnage pour réduire la résolution de l'image comme dans [22]. Nous avons aussi la partie pour l'apprentissage résiduel ainsi que la normalisation par batch entre la couche convolutive et la fonction non linéare ReLU.

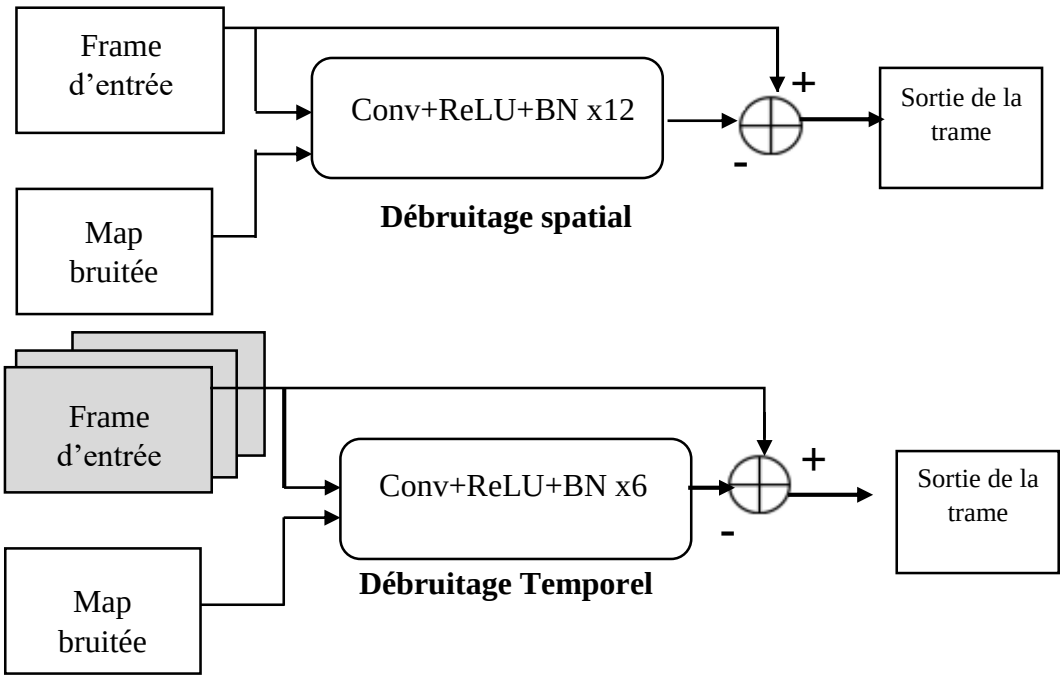


Figure 3. 29 : Architecture de bloc de débruitage spatial et temporel.

Nous avons testé DVDNet sous Windows avec Anaconda 3 à l'aide de l'invite de commande Prompte Anaconda 3. Exactement de la même manière que pour DnCNN et FFDnet, pour l'exécution de DVDnet, nous avons à la fois testé la séquence utilisée dans [34] avec la taille des séquences 960×450 ainsi que notre propre séquence.

Figure 3.30 : Débruitage DVDnet σ = 50 de taile 960x450 par DVDnet.

La séquence débruitée est considérablement améliorée. Les différents résultats sont traduits dans le tableau suivant :

Tableau 3. 15: Résultats d'évaluation des images Sras-Cov-2 par DVDnet.

Ecart type σ	PSNR (dB)	SSIM (%)	Temps d'exécution (s)
10	**34.36**	**95.42**	**213.41**
20	**31.18**	**31.84**	**203.37**
30	**29.21**	**88.14**	**182.73**
40	**27.81**	**84.58**	**200**
50	**26.7**	**81.19**	**189**

Pour le cas de nos données de séquence utilisé, nous pouvons voir la scène en mouvement, où de la même façon, la qualité visuelle reste très bonne avec une bonne cohérence temporelle :

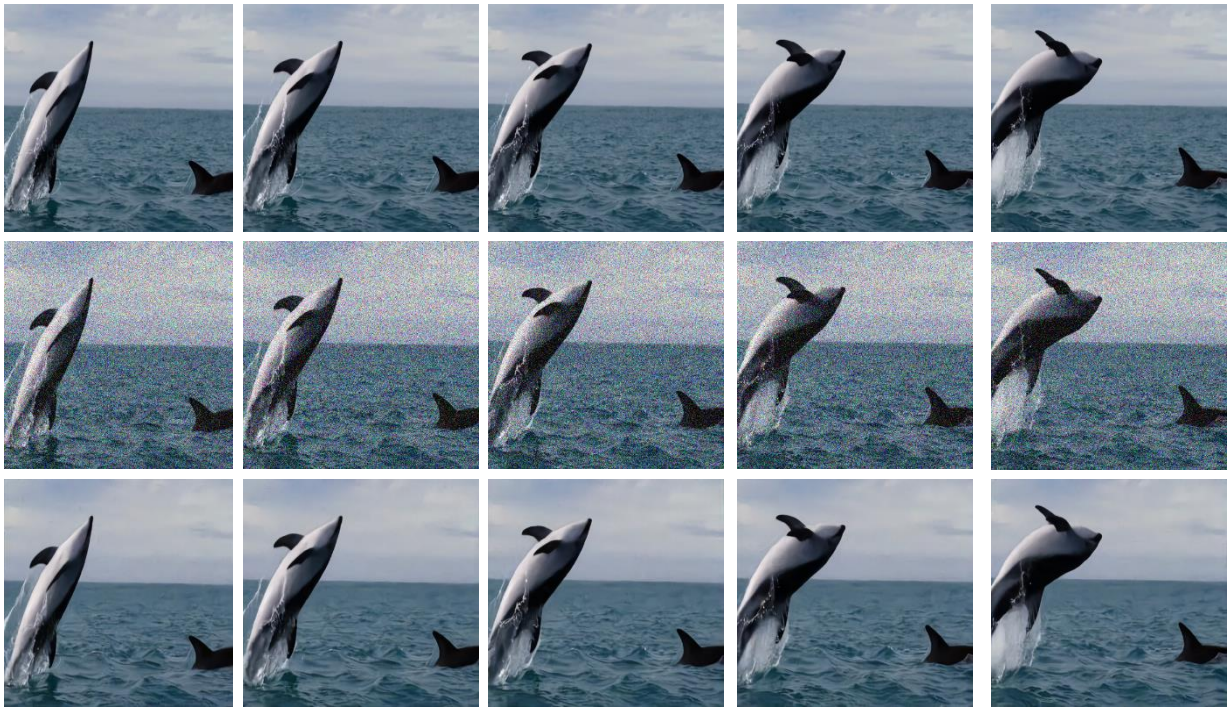


Figure 3. 31: Débruitage DVDnet de nos séquences d'images σ = 50 de taille 450x450 par DVDnet.

Nous avons donc selon le tableau des résultats ci-dessous, une bonne qualité visuelle étant donné que pour les 5 différentes valeurs de l'écart type σ, le taux reste supérieur à 90%, nous ajoutons aussi que malgré la scène en mouvement, et la fixation du plan arrière, nous remarquons que les bords de l'animal en question sur les photos restent nets.

Tableau 3. 16 : Résultats d'évaluation de nos séquences d'images par DVDnet.

Ecart type σ	PSNR (dB)	SSIM (%)	RUN (s)
10	**41**	**98.14**	**240.97**
20	**38.62**	**96**	**273.8**
30	**36.58**	**94.82**	**258.68**
40	**35.12**	**93.10**	**237.44**
50	**33.96**	**91.36**	**254.18**

3.5.2.5. FastDVDNet

FastDVDnet comme son homologue DVDnet est utilisé aussi sous Python avec comme option de plus niveau software, l'installation de Docker qui aide à utiliser le code source de l'algorithme à la fois sur Linux et sur Windows. L'entraînement de celui-ci n'est pas très

différent de DVDnet à part que celui-ci implémente une architecture auto-encodeur (Figure 3.32).

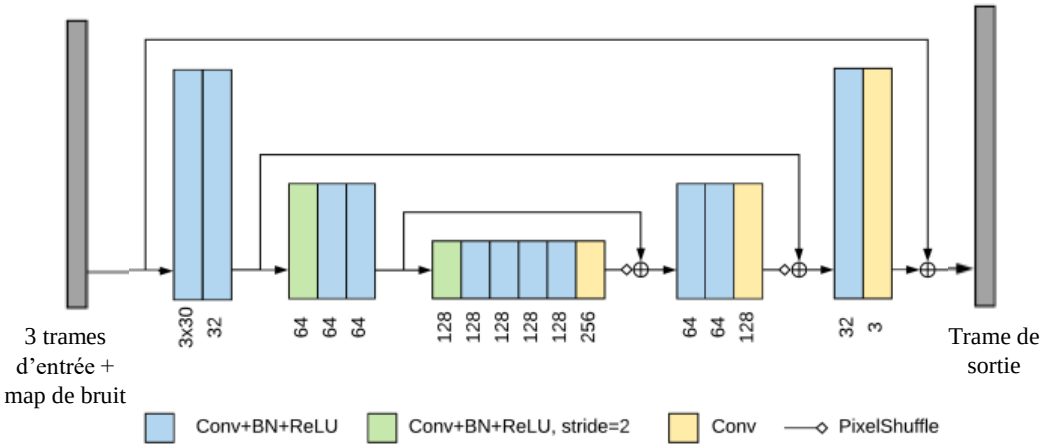


Figure 3. 32: Architecture auto-encodeur de FastDVDnet [35].

Les points qu'il faut prendre en considération à partir de cette architecture est que l'encodeur prend en considération 3 trames à l'entrée avec la carte du bruit. Implémentation des connexions résiduelles (**skip connections**) entre l'entrée et la sortie. Le sur-échantillonnage dans le décodeur est effectué avec une couche **PixelShuffel** (Figure 3.32), qui aide à réduire les artefacts de grille [36]. en utilisant un auto-encodeur avec différents blocs de débruitage (deux blocs) spatio-temporels qui partagent les mêmes poids synaptiques pour gagner plus en rapidité :

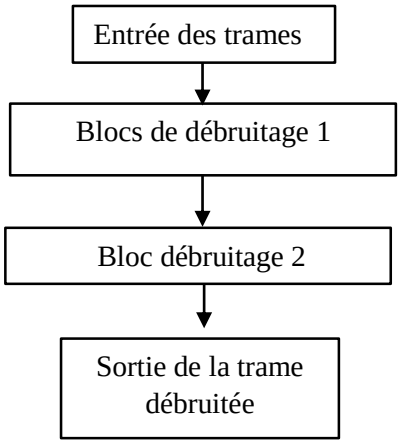


Figure 3. 33 : Organigramme de débruitage FastDVDnet.

Nous avons vu la qualité que peut procurer DVDnet, encore là, nous avons FastDVDnet qui procure aussi de très bons résultats comme nous pouvons le voir sur les résultats obtenus :

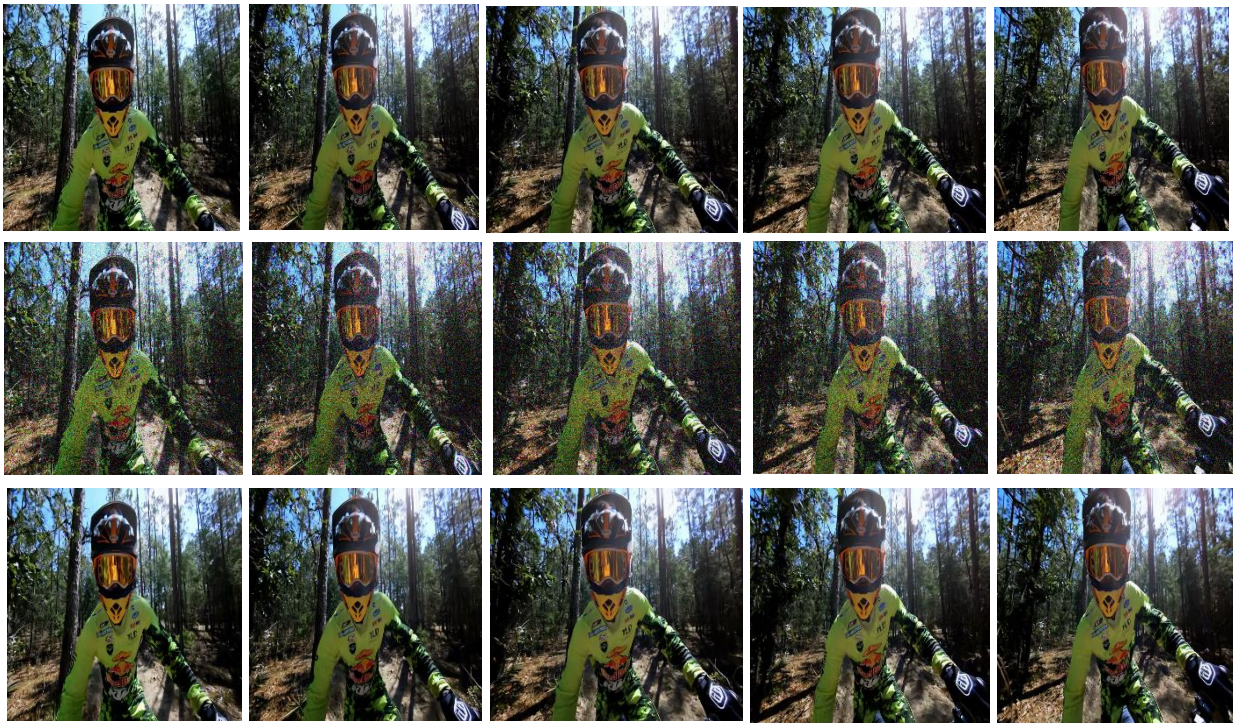


Figure 3.34: Débruitage FastDVDnet σ = 50 960x450.

Tableau 3.17: Résultats d'évaluation des séquences d'images Biker par FastDVDnet.

Ecart type σ	PSNR (dB)	SSIM (%)	Temps d'exécution (s)
10	**35.23**	**96.19**	**240**
20	**31.54**	**92.43**	**273.8**
30	**29.45**	**88.71**	**258.68**
40	**28**	**85.12**	**237.44**
50	**26.89**	**81.61**	**254.18**

Notre séquence d'images que nous avons utilisées avec une taille réduite (450×450) par rapport à la séquence de la figure 3.35 démontre selon les résultats du tableau 3.18 ci-dessous encore plus un très bon débruitage qualitatif.

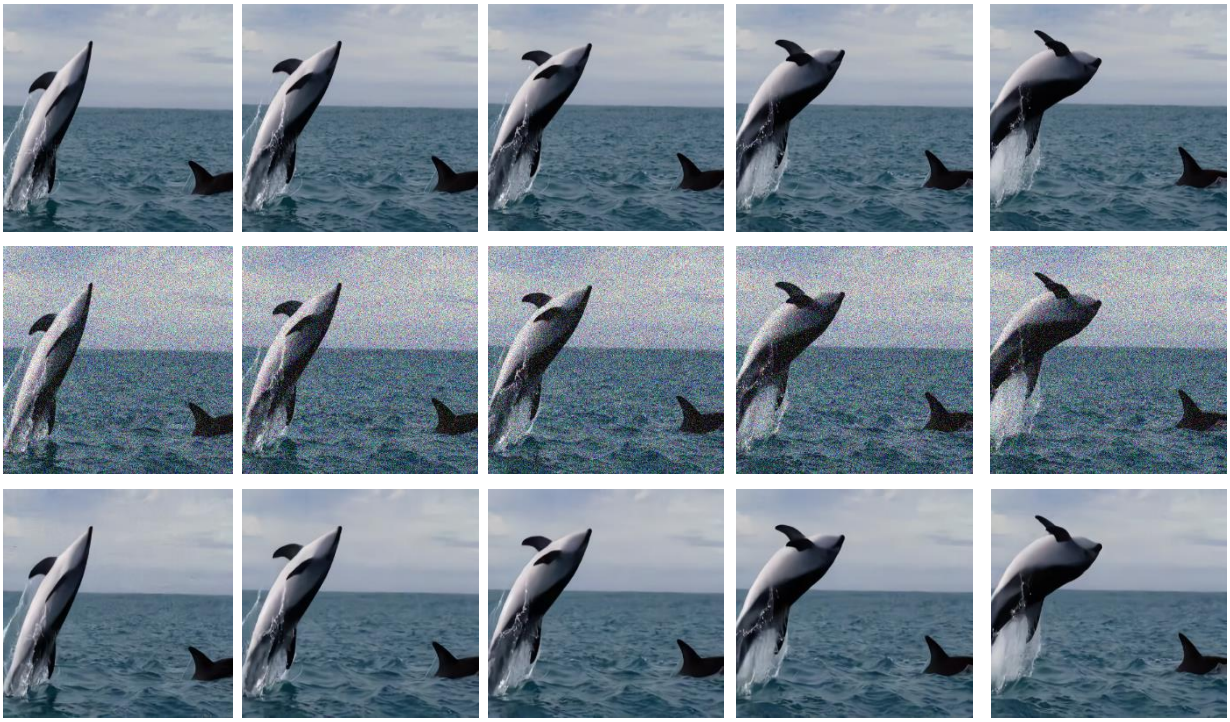


Figure 3. 35 : Débruitage de nos séquences d'images par FastDVDnet, σ=50 de taille 450x450.

Tableau 3.18: Résultats d'évaluation des images de nos séquences d'images par Fast DVDnet.

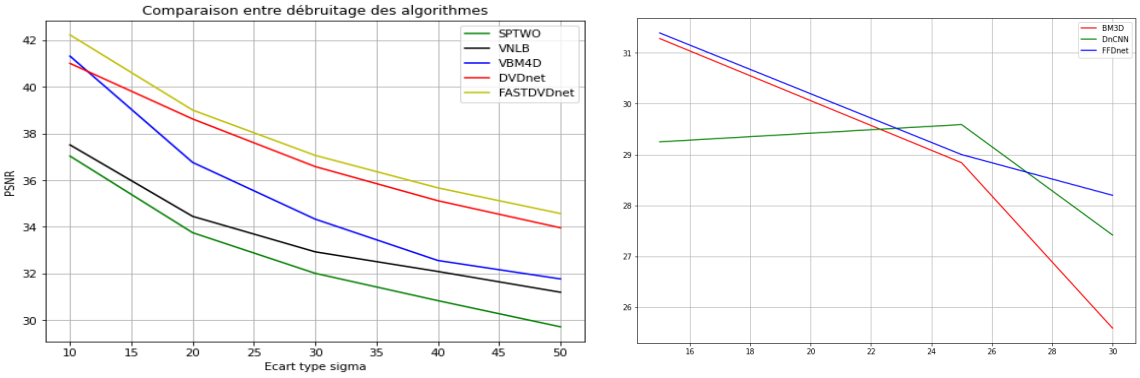
Ecart type σ	PSNR (dB)	SSIM (%)	Temps d'exécution (s)
10	**42.22**	**98.19**	**190**
20	**39**	**96.6**	**210**
30	**37.06**	**94.98**	**199**
40	**35.67**	**93.27**	**213.35**
50	**34.57**	**91.55**	**193.85**

3.6. Contribution

Dans ce travail, nous avons implémenté les algorithmes de débruitage image/vidéo les plus performants et qui se base principalement sur le formalisme Bayesien et les CNN. Nous avons établi une comparaison qualitative et quantitative où nous nous sommes concentrés sur les cinq algorithmes suscités ci-dessous notamment **DVDnet**, **FastDVDnet**, **SPTWO**, **VNLB**, **VBM4D**. Nous avons aussi comme le cas de l'algorithme SPTWO introduit le calcul du PSNR afin de rendre plus adaptable avec les autres algorithmes.

3.7. Comparaison des résultats

Pour compléter notre étude comparative, nous avons superposé les résultats de PSNR pour chaque valeur du bruit sur le même graphe.



a- Débruitage Videos. b- Débruitage Images.

Figure 3. 36: Graphique de Comparaison de débruitage entre les algorithmes Image/Vidéo en termes de PSNR.

À travers ces ceux graphes, nous remarquons que sur plusieurs valeurs de l'écart type σ, nous avons l'algorithme FastDVDnet qui donne les meilleurs résultats suivi de près par l'algorithme DVDnet, ce qui prouve qu'un réseau neuronal CNN donne de très bons résultats par rapport à une technique de débruitage classique. Sachant aussi que l'algorithme FastDVDnet possède une architecture à Auto-encodeur comme celle dont nous avons opté pour notre structure cités plus haut. De même pour le cas du débruitage image où nous avons DnCNN et FFDnet prenant une avance sur l'algorithme de débruitage classique BM3D.

En ce qui concerce le temps d'exécution on a une rapidité remarqué par BM3D et FFDnet en quelque petites secondes par rapport à l'algorithme DnCNN qui prends plusieurs dizaines de secondes à l'exécution selon les résultats des tableaux précédents. Pour le débruitage des vidéos, nous avons l'algorithme Fast DVDnet qui est plus rapide que les autres algorithmes avec une avance considérable par rapport à l'algorithme VBM4D par exemple. suivi après par DVDnet. Nous pouvons donc dire que les deux algorithmes CNN sont performant en termes de qualité de débruitage et de rapidité d'exécition.

3.8. Limites

Nous ajoutons un point important sur le temps d'exécution pour l'atténuation du bruit, où nous avons remarqué un temps d'exécution qui prenait plus de temps pour certain algorithmes que d'autres comme par exemple l'algorithme SPTWO et VNLB qui prenaient un temps allant jusqu'à plusieurs minutes pour débruiter une même séquence que pour FastDVDnet ou DVDnet qui eux prenait que quelques dizaines de secondes et VBM4D qui prenait des centaines de secondes. Il faut aussi prendre cela en raison de la machine avec laquelle on test ces résultats car l'exécution sous une GPU reste très rapide par rapport à une exécution avec CPU. Parfois même la machine risque de donner une latence très lente.

3.9. Conclusion

Nous concluons ce travail par le fait d'avoir testé plusieurs algorithmes et voir la qualité visuelle que peut procurer chaque technique sous différents plateformes ou logiciels. Nous avons pu voir dans ce chapitre la différence entre un algorithme de débruitage classique et un autre utilisant un réseau neuronal convolutif pouvant donner des résultats remarquables. Notre objectif qui était l'implémentation des algorithmes de débruitage les plus performants a été atteint avec succès. L'étude comparative effectuée est justifiée par le calcul des critères d'évaluation les plus usuels. L'intérêt de l'utilisation des CNN est bien dégagé.

Conclusion Générale et Perspective

Nous avons pu voir dans ce modeste travail l'utilisation et l'implémentation d'un apprentissage profond Deep Learning par réseau de neurones convolutive CNN pour l'atténuation de bruits additif blanc Gaussien où nous avons obtenu des résultats satisfaisants. Nous avons montré que la structure du débruitage par CNN est très performante facilitant aussi les calculs mathématiques.

Ce travail nous a permis de découvrir les différents algorithmes de débruitage que ce soit pour les images ou pour les vidéos en faisant la comparaison entre la technique classique et la technique pour apprentissage automatique, plus précisément, nous avons considéré les algorithmes **BM3D, DnCNN, VBM4D, VNLB, FFDnet, DVDnet, FastDVDnet,**

Ce thème, nous a aussi donné un avant-goût sur le domaine de la recherche scientifique au niveau du débruitage pour les systèmes embarqués et systèmes à temps réels. Nous avons aussi pu voir le débruitage des vidéos qui est un domaine peu exploité, c'est pourquoi nous pouvons proposer d'autres structures avec d'autres architectures comme par exemple le cas du réseau de neurone récurrent RNN.

Nous avons pu apprendre à travers le passage des chapitres suscités les outils qui constituent un réseau de neurones, par conséquent, le traitement de ces outils est primordial que ce soit pour le traitement des données, l'initialisation des paramètres, l'importance de la normalisation des données. Comme nous avons aussi pût voir l'enrichissement et l'apport apporté par l'ajout des techniques à l'intérieur d'un réseau de neurone comme la normalisation par batch et l'apprentissage résiduel, pour apporter et ajouter de bonnes performances.

Néanmoins, nous avons rencontré des difficultés lors de notre travail dans le côté hardware où nous étions limités en termes de performances de la machine à traiter de manière efficace l'exécution des algorithmes en raison du manque manque d'une carte GPU qui pouvait nous donner accès à certains autres algorithmes de CNN comme VNLnet malheureusement. Il est recommandé d'utiliser un calculateur incluant une GPU pour traiter plus rapidement et efficacement l'exécution. Malgré cela nous avons pu acquérir des résultats de débruitage très satisfaisants et cela est traduit par le calcul des critères d'évaluation PSNR, SSIM, MSE, et RMSE.

Bibliographie

[1] C.A Pickover, "*La fabuleuse histoire de l'intelligence artificielle Des automates aux robots humanoïdes*" *Soins*, vol. 66, no. 855. p. 9, 2021, doi: 10.1016/s0038-0814(21)00119-5.

[2] P. Kim, *Matlab deep learning: With machine learning, neural networks and artificial intelligence*. 2017.

[3] G. Saint-Cirgue, "*Apprendre le Machine Learning en une semaine*"Tous droits réservés © 2019 Guillaume Saint-Cirgue machinelearnia.com 1," 2019.

[4] G. Keifer and F. Effenberger, *Big Data Et Machine Learning*, vol. 6, no. 11. 1967.

[5] A. Chaudhary, K. S. Chouhan, J. Gajrani, and B. Sharma, *Deep Learning With PyTorch*. 2020.

[6] J. J. Heckman, R. Pinto, and P. A. Savelyev, "*Artificial Intelligence, IOT and Machine Learning*" *Angew. Chemie Int. Ed. 6(11), 951–952.*, 1967.

[7] C-A. Azencott, *Introduction au Machine Learning, Dunod, France. 2018.*

[8] H. Kinsley and D. Kukieła, "Neural Networks from Scratch in Python," p. 658, 2020.

[9] R. Rakotomalala, "Ricco Rakotomalala Tutoriels Tanagra-http://tutoriels-data-mining.blogspot.fr/ Perceptrons simples et multicouches," [Online]. Available: http://tutoriels-data-mining.blogspot.fr/.2013

[10] A. Romero and A. Romero, "Assisting the training of deep neural networks with applications to computer vision Assisting the training of deep neural networks with applications to computer vision."University of Barcelona. 2015

[11] T. Okatani, *Python Deep Learning 2nd*, vol. 33, no. 2. 2015. Packt

[12] S. Kojouharov, "Cheat Sheets for AI, Neural Networks, Machine Learning, Deep Learning and Big Data," *Becom. Hum. Explor. AI*, pp. 1–31, 2017, [Online]. Available: https://becominghuman.ai/cheat-sheets-for-ai-neural-networks-machine-learning-deep-learning-big-data-678c51b4b463.

[13] Y .Le Cun "A theoretical framework for Back-Propagation "G.Hinton, and T.Sejnowski, editors, Proceedings of the 1988 Connectionist Models Summer School p 21-28, CMU, Pittsburgh . 1988

[14] H. Habibi Aghdam, E. Jahani Heravi, and S. I. P. AG, *Guide to Convolutional Neural Networks A Practical Application to Traffic-Sign Detection and Classification*. 2018.

[15] G. Piccioli, E. De Micheli, P. Parodi, and M. Campani, "Robust method for road sign detection and recognition," *Image Vis. Comput.*, vol. 14, no. 3, pp. 209–223, 1996, doi: 10.1016/0262-8856(95)01057-2.

[16] H. Singh, *Practical Machine Learning and Image Processing For Facial Recognition, Object Detection, and Pattern Recognition Using Python-Himanshu Singh*. Apress, India

2019.

[17] A. Anwar, “Machine Learning/Data Science Interview Cheat sheets,” [Online]. Available: https://www.cheatsheets.aqeel-anwar.com.

[18] B. G. S. Ramesh Jain, Rangachar Kasturi, “MACHINE VISION Ramesh Jain,Rangachar Kasturi,Brian G.Schunck.”.University of Illinois, 1995.

[19] M. Bertalmío Editor, *Denoising of Photographic Images and Video Fundamentals, Open Challenges and New Trends*. Springer, UK, 2018.

[20] S. Escalera, *Inpainting and Denoising Challenges*. Springer, London, 2019.

[21] A. Vyas, S. Yu, and J. Paik, *Fundamentals of digital image processing*. Wiley-Blackwell, 2018.

[22] K. Dabov, A. Foi, V. Katkovnik, and K. Egiazarian, “Image denoising by sparse 3-D transform-domain collaborative filtering,” *IEEE Trans. Image Process.*, vol. 16, no. 8, pp. 2080–2095, 2007, doi: 10.1109/TIP.2007.901238.

[23] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, vol. 2016-Decem, pp. 770–778, 2016, doi: 10.1109/CVPR.2016.90.

[24] K. Zhang, W. Zuo, Y. Chen, D. Meng, and L. Zhang, “Beyond a Gaussian denoiser: Residual learning of deep CNN for image denoising,” *IEEE Trans. Image Process.*, vol. 26, no. 7, pp. 3142–3155, 2017, doi: 10.1109/TIP.2017.2662206.

[25] S. Ioffe and C. Szegedy, “Batch normalization: Accelerating deep network training by reducing internal covariate shift,” *32nd Int. Conf. Mach. Learn. ICML 2015*, vol. 1, pp. 448–456, 2015.

[26] M. Tassano, J. Delon, and T. Veit, “An analysis and implementation of the FFDNet image denoising method,” *Image Process. Line*, vol. 9, pp. 1–25, 2019, doi: 10.5201/ipol.2019.231.

[27] G. Mentzas, D. Apostolou, A. Abecker, R. Young, and C. Fyfe, *Machine Learning for Arduino, Image, and Video Analysis, Springer, London. 2008*.

[28] J. Gibson and O. Marques, *Optical Flow and Trajectory Estimation Methods*, no. 9783319449401. Springer, UK, 2016.

[29] A. Buades and J.-L. Lisani, “Video Denoising with Optical Flow Estimation,” Image Processing On Line , vol. 8, pp. 142–166, Jul. 2018.

[30] M. Maggioni, G. Boracchi, A. Foi, and K. Egiazarian, “Video denoising, deblocking, and enhancement through separable 4-D nonlocal spatiotemporal transforms,” *IEEE Trans. Image Process.*, vol. 21, no. 9, pp. 3952–3966, 2012, doi: 10.1109/TIP.2012.2199324.

[31] J. Wang, Y. Guo, Y. Ying, Y. Liu, and Q. Peng, “Fast non-local algorithm for image denoising,” *Proc. - Int. Conf. Image Process. ICIP*, no. 0, pp. 1429–1432, 2006, doi: 10.1109/ICIP.2006.312698.

[32] M. Lebrun, A. Buades, and J.-M. Morel, “Implementation of the ‘Non-Local Bayes’ (NL-Bayes) Image Denoising Algorithm,” *Image Process. Line*, vol. 3, pp. 1–42, 2013, doi: 10.5201/ipol.2013.16.

[33] P. Arias and J. M. Morel, “Video Denoising via Empirical Bayesian Estimation of Space-Time Patches,” *J. Math. Imaging Vis.*, vol. 60, no. 1, pp. 70–93, 2018, doi: 10.1007/s10851-017-0742-4.

[34] M. Tassano *et al.*, “DVDnet : Un réseau profond rapide pour le débruitage vidéo To cite this version : HAL Id : hal-02274077 DVDnet : Un réseau profond rapide pour le débruitage vidéo,” 2019.

[35] M. Tassano, J. Delon, and T. Veit, “FastDVDNet: Towards real-time deep video denoising without flow estimation,” *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, pp. 1351–1360, 2020, doi: 10.1109/CVPR42600.2020.00143.

[36] W. Shi *et al.*, “Real-Time Single Image and Video Super-Resolution Using an Efficient Sub-Pixel Convolutional Neural Network,” *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, vol. 2016-December, no. September, pp. 1874–1883, 2016, doi: 10.1109/CVPR.2016.207.

[37] A. Buades and J. L. Lisani, “Video denoising with optical flow estimation,” *Image Process. Line*, vol. 8, pp. 142–166, 2018, doi: 10.5201/ipol.2018.224.

Abstract

Nowadays, Machine Learning has proven itself as a contender for modern researches on the technical domain. We propose in this manuscript to exploit a certain field from machine learning, which is the deep learning. With the Deep Learning we'll do a study about image and Video denoising through deep neural network especially CNN (**C**onvolutional **N**eural **N**etwork) according to several recent algorithms namely: **DVDnet, FastDVDnet, FFDnet, DnCNN**. We shall compare them with classical Image/Video denoising algorithms: **BM3D**, **VBM4D**, **VNLB**, **SPTWO**. By this comparative study, we will use an evaluation criterion by calculating **PSNR**, **SSIM**, **MSE**, and **RMSE** in addition to running time and visual quality of the obtained denoised images/videos.

Keywords: Deep Learning, Denoising, Deep neural network, CNN, PSNR, SSIM

Résumé

De nos jours, l'apprentissage automatique s'est prouvé en tant que prétendant pour les recherches moderne dans le domaine technique. Nous proposons dans ce manuscrit d'exploiter un certain champ à partir de l'apprentissage automatique, qui est l'apprentissage profond. Avec l'apprentissage profond nous allons étudier sur le débruitage d'images et Vidéos à travers le réseau de neurones profond plus spécialement CNN (Réseau de neurones convolutifs) selon plusieurs algorithmes récents tel que : **DVDnet, FastDVDnet, FFDnet, DnCNN.** Nous allons les comparer avec d'autres algorithmes classiques de débruitage Image/Vidéo comme **BM3D, VBM4D, VNLB, SPTWO.** Par cette étude comparative, nous allons utiliser des critères d'évaluation par le calcul du **PSNR, SSIM, MSE et RMSE,** ajouté à cela le temps d'exécution et la qualité visuelle de l'image/video obtenu.

Mots-clés : Apprentissage profond, Débruitage, réseau de neurones profond, CNN, PSNR, SSIM

ملخص

في وقتنا الحاضر، استطاع التعلم الآلي أن يثبت نفسه كمتظاهر للبحوثات العصرية في المجال التقني. نقترح في هذه النسخة بإستغلال حقل من التعلم الآلي، الذي يتمثل في التعلم العميق. مع هذا التعلم العميق سنقوم بدراسة لتقليل الضوضاء للصور و الفديوهات من خلال الشبكة العصبية العميقة و بالخصوص CNN (الشبكة العصبية التلافيفية) بواسط العديد من الخوارزميات العصرية مثل: **DVDnet, FastDVDnet, FFDnet, DnCNN**. سنقارنهم مع خوارزميات كلاسيكية أخرى للتقليل من الضوضاء لصور/فديوهات مثل **BM3D, VBM4D, VNLB, SPTWO**. بهذه الدراسة المقارنة، سنستعمل معايير التقييم بحساب **PSNR ، SSIM، MSE وRMSE .** اضافتا الى هذا سرعة التنفيذ و النوعية البصرية للصورة و الفيديو.

كلمات مفتاحية: التعلم العميق، تقليل الضوضاء، شبكة عصبية عميقة، CNN, PSNR, SSIM.