Tabulation de l'efficacité des méthodes de débruitage d'images

Mohamed Mohamed El Bechir
Encadrant: Saïd LADJAL

Contents

1 Introduction

L'objectif de ce projet est d'évaluer et de comparer plusieurs méthodes de débruitage d'images en considérant deux aspects complémentaires : la qualité de la restauration mesurée par le PSNR, et le coût opérationnel estimé en nombre d'opérations ou en MACs par pixel. L'idée est de confronter des méthodes classiques et des méthodes récentes basées sur des réseaux de neurones, afin de les placer sur un même plan : celui de l'efficacité.

Nous avons travaillé sur le dataset CBSD68, composé de 68 images naturelles en couleur de tailles variées. Nous avons ajouté un bruit gaussien avec trois niveaux distincts ($\sigma = 5, 10, 25$), puis appliqué quatre méthodes de débruitage :

- deux méthodes classiques : BM3D et TV (algorithme de Chambolle),
- deux méthodes par réseaux de neurones : FFDNet (implémentation IPOL) et DRUNet (implémentation `deepinv`).

Nous avons organisé le travail en deux volets complémentaires. D'une part, une analyse expérimentale avec le calcul du PSNR moyen pour chaque méthode et chaque niveau de bruit. D'autre part, une analyse théorique et numérique du coût opérationnel, menée selon deux approches :

1. une dérivation analytique, où nous avons détaillé pas à pas le nombre d'opérations en fonction des briques de calcul de chaque algorithme ;
2. une estimation numérique, en particulier pour les réseaux neuronaux, où nous avons combiné le comptage manuel des convolutions et l'utilisation de la bibliothèque `ptflops`.

Pour structurer le projet, nous avons utilisé deux notebooks distincts : l'un dédié aux expériences et au calcul des PSNR, et l'autre à l'estimation des coûts. Dans ce rapport, nous rassemblons l'ensemble de ces éléments en gardant les détails mathématiques et techniques nécessaires pour assurer la traçabilité et la justification de chaque étape.

2 Données, métrique et protocole

Jeu de données

Nous avons utilisé le dataset **CBSD68**, qui contient 68 images naturelles en couleur de tailles variées. C'est un jeu de référence largement employé pour évaluer les méthodes de débruitage, car il présente à la fois de la diversité (textures fines, structures géométriques, dégradés lisses) et une taille raisonnable permettant de réaliser des tests systématiques.

Bruit simulé

Nous avons considéré un bruit gaussien additif de moyenne nulle et de variance σ^2. Trois niveaux de bruit ont été générés afin d'évaluer la robustesse des méthodes :

$$\sigma \in \{5,\ 10,\ 25\}.$$

Pour chaque image du dataset, nous avons produit une version bruitée correspondant à ces valeurs de σ.

Métrique de qualité

La performance de chaque méthode est évaluée à l'aide du **Peak Signal-to-Noise Ratio (PSNR)**. Pour une image originale x et son estimation débruitée $\hat{x}$, le PSNR est défini par :

$$\mathrm{PSNR}(x, \hat{x}) = 10 \cdot \log_{10}\left(\frac{MAX^2}{\mathrm{MSE}(x, \hat{x})}\right),$$

où MAX désigne la valeur maximale possible d'un pixel (255 pour des images codées sur 8 bits), et

$$\mathrm{MSE}(x, \hat{x}) = \frac{1}{HW}\sum_{i=1}^{H}\sum_{j=1}^{W}\left(x_{i,j} - \hat{x}_{i,j}\right)^2$$

est l'erreur quadratique moyenne entre x et $\hat{x}$.

Nous avons calculé le PSNR image par image puis pris la moyenne sur l'ensemble du dataset CBSD68 pour chaque couple (méthode, niveau de bruit). Cette valeur moyenne constitue l'indicateur de qualité que nous rapportons.

Protocole expérimental

Le protocole suivi est le suivant :

1. Générer les images bruitées à partir de CBSD68 pour $\sigma = 5, 10, 25$.
2. Appliquer chacune des quatre méthodes de débruitage (BM3D, TV, FFDNet, DRUNet).
3. Calculer le PSNR de sortie pour chaque image.
4. Moyenne sur l'ensemble du dataset afin d'obtenir le PSNR moyen par méthode et par niveau de bruit.

Ce protocole permet de comparer les méthodes de manière équitable, dans des conditions strictement identiques.

3 BM3D

La méthode **BM3D** (*Block-Matching and 3D Filtering*) est aujourd'hui l'un des algorithmes de débruitage classiques les plus performants. Son principe repose sur l'exploitation de la redondance non-locale dans l'image. Plutôt que de traiter les pixels de façon isolée, BM3D regroupe des patches (petits blocs carrés de taille $k \times k$) présentant une forte similarité, même s'ils sont éloignés spatialement.

Pour chaque patch de référence, l'algorithme recherche dans une fenêtre de taille $n \times n$ les L patches les plus proches selon une distance quadratique normalisée. Ces patches sont empilés pour former un bloc 3D.

Une fois le groupement réalisé, le bloc est transformé par :

- une transformée 2D appliquée indépendamment sur chaque patch (ondelettes biorthogonales Bior1.5 lors de la première passe, DCT lors de la seconde),
- une transformée 1D de Hadamard appliquée le long de la troisième dimension (axe des patches).

Dans ce domaine transformé, un filtrage collaboratif est effectué :

- lors de la première passe, un seuillage dur des coefficients permet d'obtenir une estimation basique de l'image ;
- lors de la seconde passe, un filtrage de Wiener affine cette estimation en exploitant à la fois l'image bruyante et l'image basique.

Les patches filtrés sont ensuite ramenés dans le domaine spatial par l'inverse des transformées, puis replacés dans l'image. Comme un pixel appartient en général à plusieurs patches différents, une étape d'agrégation pondérée (fenêtre de Kaiser, normalisation par des poids) est appliquée pour combiner toutes les contributions et reconstruire l'image finale.

L'algorithme BM3D repose donc sur trois piliers : (i) le **block-matching** (sélection de patches similaires), (ii) le **filtrage collaboratif** dans un domaine transformé, et (iii) l'**agrégation** des estimations partielles. Les paramètres principaux sont la taille des patches k, la taille de la fenêtre n, le nombre de patches groupés L (différent pour les passes Hard et Wiener), ainsi que le stride p qui contrôle la densité des patches de référence.

Analyse du coût opérationnel

Hypothèses et notations. On note $H \times W$ la taille de l'image ($N = HW$ pixels), k la taille du patch ($k \times k$), n la taille de la fenêtre de recherche ($n \times n$), M le nombre de candidats considérés par patch de référence (en pratique $M \simeq (n-k+1)^2$ et $M \approx n^2$ si $n \gg k$), L la taille du groupe (distincte selon la passe : L_{hard} à l'étape 1 et L_{wien} à l'étape 2), et p le pas (stride) entre patches de référence. *Convention de comptage :* chaque addition, soustraction, multiplication, division ou comparaison compte pour 1 opération. Les accès mémoire et l'usage de constantes pré-tabulées (p. ex. cosinus de la DCT) ne sont pas comptés à chaque occurrence.

Modèle DCT 1D. On modélise une DCT 1D rapide de longueur k par

$$c_{1\text{D}}^{\text{DCT}}(k) \;\approx\; (\alpha_{\text{M}} + \alpha_{\text{A}})\, k \log_2 k,$$

où $(\alpha_{\text{M}}, \alpha_{\text{A}})$ sont des constantes empiriques (multiplications / additions). Pour rester prudent, on retient $(\alpha_{\text{M}}, \alpha_{\text{A}}) = (2, 4)$, soit $c_{1\text{D}}^{\text{DCT}}(k) \approx 6k \log_2 k$.

(1) Groupage (block-matching). La distance L^2 normalisée entre deux patches $P, Q \in \mathbb{R}^{k \times k}$ est

$$d(P, Q) \;=\; \frac{1}{k^2} \sum_x (P_x - Q_x)^2.$$

Pour un candidat donné : k^2 soustractions, k^2 multiplications (carrés), k^2 additions (accumulation), soit $3k^2$ opérations, puis 1 division et 1 comparaison (test de seuil). Ainsi, pour M candidats :

$$\underbrace{M\,(3k^2+2)}_{\text{distances + seuil}}\,.$$

La sélection des L meilleurs candidats peut être modélisée par un tri complet ($M \log_2 M$ comparaisons) ou, plus finement, par un tas borné à L ($M \log_2 L$). En modèle simple (tri complet) :

$$\boxed{C_{\mathrm{BM}}(k,n,M) \;\approx\; M\,(3k^2+2) \;+\; M\log_2 M.} \tag{1}$$

(2) Transformées (par patch, puis sur l'axe des groupes).

(a) DCT 2D séparable (passe 2). Une DCT 2D séparable $k \times k$ équivaut à k DCT 1D sur les lignes puis k DCT 1D sur les colonnes, soit $2k$ DCT 1D :

$$T_{2D}^{\mathrm{DCT}}(k) \;=\; 2k\, c_{1\mathrm{D}}^{\mathrm{DCT}}(k) \;=\; 2(\alpha_{\mathrm{M}}+\alpha_{\mathrm{A}})\, k^2 \log_2 k.$$

Avec $(2,4)$, on obtient $T_{2D}^{\mathrm{DCT}}(k) \approx 12\, k^2 \log_2 k$. On notera $T_{2D}^{\mathrm{DCT}}(k) = \beta\, k^2 \log_2 k$ avec $\beta \in [10,14]$ pour couvrir les variantes d'implémentation ; aller et retour ont des coûts similaires.

(b) Ondelette biorthogonale Bior1.5 2D (passe 1). L'analyse/synthèse biorthogonale 2D (filtres 1D longueur 10) donne, par axe :

$$\text{Analyse } \approx 11k^2, \quad \text{Synthèse } \approx 19k^2,$$

soit sur deux axes (2D) : $22k^2$ à l'aller et $38k^2$ au retour. Pour un aller-retour :

$$\boxed{T_{2D,\mathrm{fwd+inv}}^{\mathrm{Bior}}(k) \;\approx\; 60\,k^2.} \tag{2}$$

(c) Walsh–Hadamard 1D (axe des groupes). La transformée de Hadamard de longueur L se calcule via des papillons somme/différence. Le coût de l'aller est $L \log_2 L$ additions, l'inverse idem. Comme il existe k^2 coefficients 2D par patch :

$$\boxed{T_{3\mathrm{D}}^{\mathrm{Had}}(k,L) \;=\; 2\,k^2\,L\log_2 L.} \tag{3}$$

(3) Non-linéarités et pondérations.

(a) Seuillage dur (passe 1). Une comparaison par coefficient :

$$\boxed{C_{\text{hard-thresh}} \;=\; k^2\, L_{\mathrm{hard}}.} \tag{4}$$

Le poids de groupe $w_P^{\mathrm{hard}} = 1/N_P$ ajoute 1 division par groupe (négigeable à l'échelle des $k^2 L$ coefficients).

(b) Filtre de Wiener (passe 2). Pour chaque coefficient, on calcule

$$\omega \;=\; \frac{|A|^2}{|A|^2+\sigma^2} \quad \text{et} \quad Y_{\text{filtré}} = \omega X,$$

ce qui représente (par coefficient) : 1 multiplication (carré), 1 addition (dénominateur), 1 division (rapport), 1 multiplication (application), soit 4 opérations :

$$\boxed{C_{\mathrm{Wiener}} \;=\; 4\,k^2\,L_{\mathrm{wien}}.} \tag{5}$$

Le poids $w_P^{\mathrm{wien}} = \|\omega\|_2^{-2}$ requiert l'accumulation de ω^2 (1 multiplication + 1 addition par coefficient) puis 1 division :

$$\boxed{C_{\|\omega\|^{-2}} \;=\; 2\,k^2\,L_{\mathrm{wien}} + 1.} \tag{6}$$

(4) Agrégation. À la recomposition, pour chaque pixel de patch : application de la fenêtre (1 multiplication), ajout pondéré au numérateur (1 multiplication + 1 addition), et au dénominateur (1 addition) :

$$4 \text{ opérations / pixel de patch } \Rightarrow\ 4k^2 \text{ par patch } \Rightarrow\ \boxed{4k^2 L \text{ par groupe.}}$$

(5) Coût par étape (par patch de référence).

Étape 1 — seuillage dur.

$$\begin{aligned} C_{\text{step1}} \approx\ & \underbrace{C_{\text{BM}}(k,n,M)}_{\text{groupage, (1)}} + \underbrace{60k^2\, L_{\text{hard}}}_{\text{Bior 2D aller+retour}} + \underbrace{2k^2\, L_{\text{hard}} \log_2 L_{\text{hard}}}_{\text{Hadamard aller+retour}} + \underbrace{k^2\, L_{\text{hard}}}_{\text{seuillage}} \\ & + \underbrace{4k^2\, L_{\text{hard}}}_{\text{agrégation}} + \underbrace{1}_{\text{inversion du poids}} \end{aligned} \tag{7}$$

$$= C_{\text{BM}}(k,n,M) + \underbrace{k^2\, L_{\text{hard}}(60 + 2\log_2 L_{\text{hard}} + 1 + 4)}_{\substack{\text{Bior 2D A/R} \\ \text{+ Hadamard A/R} \\ \text{+ seuillage + agrégation}}} + 1 \tag{8}$$

$$= C_{\text{BM}}(k,n,M) + k^2\, L_{\text{hard}}(65 + 2\log_2 L_{\text{hard}}) + 1. \tag{9}$$

Étape 2 — filtrage Wiener. La passe Wiener effectue le groupage sur l'estimation basique de l'étape 1, mais filtre le bloc construit depuis l'image bruyante ; on applique deux DCT 2D directes (basique + bruyante) et une DCT inverse (bruyante), ainsi que trois Hadamard (aller basique + aller bruyante + retour bruyante) :

$$\begin{aligned} C_{\text{step2}} \approx\ & \underbrace{C_{\text{BM}}(k,n,M)}_{\text{groupage, (1)}} + \underbrace{(2+1)\, T_{2D}^{\text{DCT}}(k)\, L_{\text{wien}}}_{\text{2 aller + 1 retour}} + \underbrace{3k^2\, L_{\text{wien}} \log_2 L_{\text{wien}}}_{\text{Had. aller (b., n.) + retour (n.)}} \\ & + \underbrace{4k^2\, L_{\text{wien}}}_{\text{Wiener}} + \underbrace{(2k^2\, L_{\text{wien}} + 1)}_{\text{poids } \|\omega\|_2^{-2}} + \underbrace{4k^2\, L_{\text{wien}}}_{\text{agrégation}} \end{aligned} \tag{10}$$

$$= C_{\text{BM}}(k,n,M) + 3\, T_{2D}^{\text{DCT}}(k)\, L_{\text{wien}} + \underbrace{k^2\, L_{\text{wien}}(3\log_2 L_{\text{wien}} + 8 + 2)}_{\substack{\text{Had. A/R} \\ \text{+ Wiener + agrégation + poids}}} + 1 \tag{11}$$

$$= C_{\text{BM}}(k,n,M) + 3\, T_{2D}^{\text{DCT}}(k)\, L_{\text{wien}} + k^2\, L_{\text{wien}}(10 + 3\log_2 L_{\text{wien}}) + 1. \tag{12}$$

(6) Coût total par pixel. On traite environ un patch de référence tous les p^2 pixels ; le nombre de patches de référence parcourus est donc $\simeq N/p^2$. En notant C_{step1} et C_{step2} les coûts *par patch de référence*, le coût ramené au pixel vaut

$$\boxed{\text{Ops/pixel} \approx \frac{C_{\text{step1}} + C_{\text{step2}}}{p^2}.} \tag{13}$$

Ancrage d'implémentation Python (`bm3d`). Le paquet Python (`pip install bm3d`) n'expose pas directement les paramètres internes de l'algorithme. Il s'appuie toutefois sur une implémentation fidèle à IPOL (Lebrun, 2012), de sorte que les valeurs par défaut sont implicitement héritées de cette référence. Ainsi, sauf indication contraire, on suppose :

$$k = 8, \quad n = 39, \quad p = 3, \quad L_{\text{hard}} = 16, \quad L_{\text{wien}} = 32,$$

et la sélection des candidats repose sur un **tri complet** des M distances (modèle $M \log_2 M$). Ces paramètres constituent la base de l'instanciation numérique dans la suite.

Instanciation numérique (opérations/pixel). Avec $k = 8$, $n = 39 \Rightarrow M = (n-k+1)^2 = (39-8+1)^2 = 32^2 = 1024$, $L_{\text{hard}} = 16$, $L_{\text{wien}} = 32$, $p = 3$, et $\beta = 12$ pour la DCT 2D, on obtient :

Groupage (par patch) :

$$C_{\text{BM}} = M(3k^2+2) + M\log_2 M = 1024\,(3\cdot 64+2) + 1024\cdot 10 = 198\,656 + 10\,240 = \mathbf{208\,896}.$$

Passe 1 ((9)) :

$$\begin{aligned} k^2 L_{\text{hard}} &= 64\cdot 16 = 1\,024, \\ 65 + 2\log_2 L_{\text{hard}} &= 65 + 2\cdot 4 = 73 \end{aligned}$$

$$\Rightarrow \boxed{C_{\text{step1}} = 208\,896 + 1\,024\cdot 73 + 1 = \mathbf{283\,649}}$$

Passe 2 ((12)) :

$$\begin{aligned} 3T_{2D}^{\text{DCT}}(8) L_{\text{wien}} &= 3\,\beta\,k^2\log_2 k = 3\cdot 12\cdot 64\cdot 3\cdot 32 = 221\,184, \\ k^2 L_{\text{wien}} &= 64\cdot 32 = 2\,048, \\ 10 + 3\log_2 L_{\text{wien}} &= 10 + 3\cdot 5 = 25. \end{aligned}$$

d'où

$$\boxed{C_{\text{step2}} = 208\,896 + 221\,184 + 2\,048\cdot 25 + 1 = \mathbf{481\,281}}.$$

Normalisation ($p = 3$) :

$$\boxed{\text{Ops/pixel} \;\approx\; \frac{283\,649 + 481\,281}{3^2} \;=\; \frac{764\,930}{9} \;\approx\; \mathbf{84\,992}\ \text{ops/pixel}.}$$

Sensibilités utiles (mêmes paramètres k, n, L, p).

- **Constante DCT.** En prenant $\beta \in [10, 14]$:

$$\text{Ops/pixel} \;\approx\; \begin{cases} \mathbf{80\,896} & (\beta = 10), \\ \mathbf{84\,992} & (\beta = 12), \\ \mathbf{89\,088} & (\beta = 14), \end{cases}$$

  soit une plage $\sim$ [81k, 89k].

- **Sélection $M\log_2 M$ vs $M\log_2 L$.** En remplaçant le tri complet par un tas borné ($M\log_2 L$), on obtient Ops/pixel $\approx$ **83 741** pour $\beta = 12$ (gain modeste car le terme $3Mk^2$ domine).

4 TV (algorithme de Chambolle)

Le débruitage par **Variation Totale** (TV) repose sur le modèle de Rudin–Osher–Fatemi (ROF) :

$$\min_u \; \frac{1}{2}\,\|u - f\|_2^2 + \lambda\,\text{TV}(u), \qquad \text{TV}(u) = \sum_{i,j} \sqrt{(\partial_x u_{i,j})^2 + (\partial_y u_{i,j})^2}.$$

Le premier terme impose la fidélité aux données bruitées f ; le terme TV favorise des solutions à bords nets (suppression du bruit tout en préservant les discontinuités). L'algorithme de **Chambolle** résout ce problème via une *formulation duale* : on met à jour un champ $p = (p_x, p_y)$ par itérations successives, puis on reconstruit

$$u^k \;=\; f - \lambda\,\text{div}\,p^k, \quad p^{k+1} \;=\; \frac{p^k + c\,\nabla u^k}{1 + c\,s^k}, \quad c = \tfrac{\tau}{\lambda} > 0,$$

où $\nabla u^k = (\partial_x u^k, \partial_y u^k)$ et

$$s^k = \begin{cases} \sqrt{(\partial_x u^k)^2 + (\partial_y u^k)^2} & \text{(isotrope)} \\ |\partial_x u^k| + |\partial_y u^k| & \text{(anisotrope).} \end{cases}$$

La divergence est calculée en différences arrière, le gradient en différences avant. Après T itérations (ou convergence), la sortie est $u^\star = f - \lambda \operatorname{div} p^T$.

Analyse du coût opérationnel

Hypothèses et conventions. On travaille *par pixel et par itération*. Chaque **addition, soustraction, multiplication, division, comparaison, racine carrée, valeur absolue** compte pour **1 opération**. Les accès mémoire et la gestion des bords sont négligés (poids relatif $\mathcal{O}(1/\min(H, W))$). Discrétisations:

$$\begin{aligned} \partial_x u(i,j) &= u(i+1,j) - u(i,j), \\ \partial_y u(i,j) &= u(i,j+1) - u(i,j), \\ \operatorname{div} p(i,j) &= [p_x(i,j) - p_x(i-1,j)] + [p_y(i,j) - p_y(i,j-1)]. \end{aligned}$$

Variante *isotrope* avec dénominateur $1 + c\,s$ (forme utilisée). On détaille, *par pixel et par itération* :

(a) **Divergence** $d = \operatorname{div} p$: 2 soustractions + 1 addition = **3 ops**.

(b) **Mise à jour** u : $u = f - \lambda d$: 1 multiplication + 1 soustraction = **2 ops**.

(c) **Gradient** ∇u: 2 soustractions = **2 ops**.

(d) **Norme isotrope** $s = \sqrt{(\partial_x u)^2 + (\partial_y u)^2}$: 2 multiplications (carrés) + 1 addition + 1 racine = **4 ops**.

(e) **Dénominateur** $d_{\text{den}} = 1 + c\,s$: 1 multiplication + 1 addition = **2 ops**.

(f) **Mise à jour duale** pour deux composantes:

$$p_x^{new} = \frac{p_x + c\,\partial_x u}{d_{\text{den}}}, \qquad p_y^{new} = \frac{p_y + c\,\partial_y u}{d_{\text{den}}}.$$

Pour *chaque* composante: 1 multiplication ($c\,\partial$) + 1 addition ($+p$) + 1 division ($/d_{\text{den}}$) = 3 ops, donc **6 ops** au total.

$\Rightarrow$ Total par itération (isotrope $1+cs$) $= 3 + 2 + 2 + 4 + 2 + 6 =$ **19** ops/pixel/itération.

Coût de sortie (après la dernière itération). On calcule $u^\star = f - \lambda \operatorname{div} p^T$: 1 divergence (**3 ops**) + 1 multiplication + 1 soustraction (**2 ops**) $\Rightarrow$

Post-traitement = **5** ops/pixel.

Formule finale (isotrope $1+cs$). Pour T itérations effectives,

$$\text{Ops/pixel} \approx 19\,T + 5.$$

Autres variantes (pour comparaison).

- **Anisotrope** $1+c(|\partial_x u|+|\partial_y u|)$: la norme vaut 2 absolus + 1 addition = **3 ops** (au lieu de 4) $\Rightarrow$ **18 ops/pixel/itération**, et toujours +5 pour la sortie.
- **Projection** $p^{k+1} = \dfrac{q}{\max(1, \|q\|)}$ avec $q = p^k + c\,\nabla u^k$:
  - isotrope : 7 (div, mise à jour u, gradient) + 4 (construction q) + 4 (norme) + 1 (max) + 2 (divisions) = **18 ops/itération**,
  - anisotrope (norme $|q_x| + |q_y|$) : **17 ops/itération**.

Ancrage d'implémentation Python (`scikit-image`). Dans `scikit-image` (`denoise_tv_chambolle`), la mise à jour est celle de la **variante isotrope** avec division par $1 + c\,s$. Les paramètres sont :

- `weight` ($= \lambda$ dans notre écriture) : intensité de la régularisation ;
- `eps` : tolérance relative sur la variation d'énergie ; `max_num_iter` : plafond d'itérations ;
- `channel_axis` pour RGB ;

Le nombre d'itérations effectif est

$$T = \min(T_\varepsilon,\ \texttt{max\_num\_iter}), \quad T_\varepsilon = \min\{k \geq 1 :\ |E_{k-1} - E_k| < \varepsilon\, E_0\},$$

où E_k est une énergie normalisée calculée à chaque itération. *C'est donc T qui pilote directement le coût*, via $19\,T + 5$.

Instanciation numérique (ops/pixel). Pour donner un ordre de grandeur reproductible, on reporte ci-dessous la valeur Ops/pixel $\approx 19\,T + 5$ pour des valeurs typiques de T (selon `eps` et `max_num_iter`) :

T	Ops/pixel (isotrope $1+cs$)
50	$19 \times 50 + 5 =$ **955**
100	$19 \times 100 + 5 =$ **1905**
200	$19 \times 200 + 5 =$ **3805**

Ces valeurs ne dépendent pas de $H \times W$ (comptage « par pixel »).

5 FFDNet (implémentation IPOL)

FFDNet est un réseau convolutif de débruitage qui repose sur deux principes clés : (i) travailler à *basse résolution par réarrangement de pixels* afin d'élargir le champ réceptif et réduire le coût de calcul, (ii) rendre le modèle *conditionnel au niveau de bruit* en introduisant une *carte de bruit* en entrée. Un unique réseau peut ainsi traiter un large éventail de niveaux σ sans réentraînement.

Données d'entraînement. FFDNet a été entraîné sur un ensemble de **1 024 000 patches** extraits de la *Waterloo Exploration Database*. Les patches ont une taille de 64×64 pour les images en niveaux de gris et 50×50 pour les images couleur. Deux variantes du modèle sont proposées : une pour les images en niveaux de gris (1 canal) et une autre pour les images couleur (3 canaux). Lors de l'entraînement, un bruit gaussien blanc additif (AWGN) est appliqué avec un écart-type σ tiré de l'intervalle $[0, 75]$, et une *carte de bruit* constante (valeurs égales à σ) est concaténée en entrée du réseau. Des opérations d'augmentation de données (recadrages aléatoires, changements d'échelle et flips) sont également utilisées afin d'accroître la diversité du jeu d'entraînement.

Entrée conditionnelle et réarrangement. Soit une image $I \in \mathbb{R}^{C\times H\times W}$ et un niveau de bruit σ (scalaire ou carte). FFDNet applique un *pixel-unshuffle* de facteur 2 qui réarrange I en quatre sous-images intercalées, puis concatène la carte de bruit. On obtient un tenseur

$$X \in \mathbb{R}^{(4C+s)\times(H/2)\times(W/2)}, \qquad s \in \{1, 4\}.$$

Le *pixel-unshuffle* est une permutation d'indices (pas de calcul arithmétique). Ainsi, l'aire spatiale est divisée par 4, et le nombre de canaux est multiplié par 4 plus le ou les canaux de bruit. La carte de bruit S sert de variable de *conditionnement* : elle guide le réseau pour adapter l'amplitude du filtrage en fonction du niveau de bruit local (constant ou spatialement variant).

Architecture convolutionnelle et sortie. Le cœur du modèle est une colonne de convolutions 3×3 à largeur constante W et stride 1 sur la grille $(H/2) \times (W/2)$:

$$(4C+s) \xrightarrow{\text{Conv } 3\times3} W \xrightarrow{D-2 \text{ blocs } (W\to W)} W \xrightarrow{\text{Conv } 3\times3} 4C.$$

Chaque couche est suivie d'une non-linéarité (ReLU). La normalisation de lots (BN), utilisée uniquement à l'entraînement, peut être absorbée à l'inférence dans les poids et biais des convolutions, sans coût supplémentaire. La profondeur D et la largeur W varient selon la variante (couleur ou niveaux de gris). Dans l'implémentation IPOL :

$$\text{couleur : } C = 3,\ D \approx 12,\ W \approx 96 \qquad \text{et} \qquad \text{N\&B : } C = 1,\ D \approx 15,\ W \approx 64.$$

La dernière couche produit $4C$ cartes internes qui sont réarrangées par un *pixel-shuffle* pour retrouver la résolution (H, W) et C canaux en sortie. Selon la variante d'entraînement, le réseau prédit soit la carte résiduelle de bruit (soustraite à l'entrée), soit directement l'image débruitée $\widehat{I} \in \mathbb{R}^{C\times H\times W}$.

Analyse du coût opérationnel

Pour les réseaux de neurones (FFDNet, DRUNet), le coût est dominé par les opérations de convolution. La métrique de référence dans la littérature consiste à compter le nombre de **MACs** (*Multiply–Accumulate*), chaque MAC correspondant à une multiplication suivie d'une addition :

$$y \leftarrow y + w \cdot x.$$

Afin de comparer des méthodes de débruitage de manière indépendante de la taille d'image $H \times W$, on normalise systématiquement par le nombre de pixels en entrée :

$$\text{MACs/pixel} = \frac{\text{nombre total de MACs sur une image}}{H\,W}.$$

Dans ce rapport, et pour rester cohérent avec l'analyse des méthodes non-apprises (BM3D, TV), nous présentons finalement les résultats en **ops/pixel**. Nous adoptons l'approximation usuelle

$$\boxed{\text{ops/pixel} \approx 2 \times \text{MACs/pixel}}$$

car un MAC regroupe en pratique une multiplication et une addition.

Brique de base : convolution 2D. Pour une convolution `Conv2d` de taille $K_h \times K_w$, entrée Cin, sortie Cout, sur une grille $H_{\text{out}} \times W_{\text{out}}$, le coût en multiplications–accumulations est :

$$\text{MACs} = H_{\text{out}} W_{\text{out}} \times \text{Cout} \times \text{Cin} \times K_h K_w.$$

Nous normalisons *par pixel d'entrée* $(H \times W)$:

$$\text{MACs/pixel} = \frac{\text{MACs}}{HW}.$$

Impact du pixel-unshuffle. FFDNet réarrange l'image $C \times H \times W$ en $(4C+s) \times H/2 \times W/2$, avec s le canal de bruit. Toutes les convolutions travaillent donc sur $H'W' = \frac{HW}{4}$ pixels. Chaque terme hérite d'un facteur $\frac{1}{4}$ dans le coût par pixel d'entrée.

Architecture convolutionnelle. Le réseau contient D couches convolutionnelles 3×3, largeur W : - Tête : $(4C + s) \to W$, - $(D - 2)$ blocs intermédiaires : $W \to W$, - Queue : $W \to 4C$.

Le coût total par pixel d'entrée est :

$$\text{MACs/pixel} = \tfrac{9}{4}\Big[\, W(4C + s) \;+\; (D-2)W^2 \;+\; 4CW \,\Big].$$

où le facteur 9 vient de la taille du noyau (3×3).

Conversion en ops/pixel. Nous adoptons la convention 1 MAC = 2 opérations (1 multiplication + 1 addition). Ainsi :

$$\boxed{\text{Ops/pixel} \;=\; 2 \times \text{MACs/pixel}}.$$

Ancrage d'implémentation (IPOL). Dans l'implémentation IPOL :

- **Couleur** $(C = 3)$: $D = 12$, $W = 96$, $s = 3$.
- **Niveaux de gris** $(C = 1)$: $D = 15$, $W = 64$, $s = 3$.

Instanciations numériques. Couleur :

$$\begin{aligned}\text{MACs/pixel} &= \tfrac{9}{4}\big[96(4\cdot 3+3) + 10\cdot 96^2 + 4\cdot 3\cdot 96\big]\\ &= \tfrac{9}{4}\big[96\cdot 15 + 10\cdot 9\,216 + 1\,152\big]\\ &= \tfrac{9}{4}\cdot 94\,080 \;=\; 213\,192.\end{aligned}$$

Donc

$$\boxed{\text{Ops/pixel} \approx 426\,384}.$$

Niveaux de gris :

$$\begin{aligned}\text{MACs/pixel} &= \tfrac{9}{4}\big[64(4\cdot 1+1) + 13\cdot 64^2 + 256\big]\\ &= \tfrac{9}{4}\big[64\cdot 5 + 13\cdot 4\,096 + 256\big]\\ &= \tfrac{9}{4}\cdot 53\,568 \;=\; 120\,528.\end{aligned}$$

Donc

$$\boxed{\text{Ops/pixel} \approx 241\,056}.$$

- Le terme dominant est $(D - 2)W^2$: le coût croît *quadratiquement* avec la largeur W, et linéairement avec la profondeur D.
- Les canaux d'entrée $(4C + s)$ et de sortie $(4C)$ jouent un rôle secondaire (effet linéaire).
- L'écart entre couleur et N&B reflète surtout la largeur ($W = 96$ vs $W = 64$) et la profondeur ($D = 12$ vs $D = 15$).

6 DRUNet

DRUNet (*Denoising Residual U-Net*) est un modèle convolutif moderne qui combine trois idées : (i) la structure en *U-Net* (encodeur–décodeur multi-échelles avec connexions de saut), (ii) la *résidualité* (le réseau prédit la carte de bruit, soustraite ensuite à l'image d'entrée), (iii) le *conditionnement par la carte de bruit*, comme dans FFDNet, pour gérer plusieurs niveaux σ avec un seul réseau.

Données d'entraînement. DRUNet est pré-entraîné sur un large ensemble d'images naturelles de la base *ImageNet*, recadrées et redimensionnées en 256×256. Pendant l'entraînement, des patches sont extraits aléatoirement et un bruit gaussien blanc additif d'écart-type σ tiré dans l'intervalle $[0, 75]$ est ajouté artificiellement. Une carte de bruit correspondante, composée de valeurs constantes égales à σ, est concaténée aux canaux d'entrée. Le modèle est appris directement en couleur (3 canaux) et, de même que pour FFDNet, ce conditionnement explicite permet de gérer de façon flexible toute une plage de niveaux de bruit avec un seul réseau.

Architecture et caractéristiques

L'architecture suit une organisation classique en U-Net :

- **Tête** : convolution 3×3 qui projette les $(C + s)$ canaux d'entrée (C canaux image, $s = 1$ canal bruit) vers W canaux internes.
- **Encodeur** : L niveaux hiérarchiques, chacun constitué de plusieurs blocs résiduels (deux convolutions 3×3 avec ReLU), suivis d'un downsampling (stride 2) qui réduit la résolution et double les canaux.
- **Bottleneck** : plusieurs blocs résiduels à la résolution la plus basse, avec un grand nombre de canaux.
- **Décodeur** : symétrique de l'encodeur, avec convolutions 3×3, upsampling par `ConvTranspose2d` (stride 2), et concaténation des features correspondants de l'encodeur (*skip connections*).
- **Queue** : convolution 3×3 ramenant W canaux à C (sortie image), interprétée comme une résiduelle de bruit soustraite à l'entrée.

Ses caractéristiques clés sont le **conditionnement explicite** par la carte de bruit, la **structure multi-échelles** typique des U-Net, et la **résidualité** qui stabilise l'apprentissage tout en préservant les détails fins.

Analyse du coût opérationnel

Pour une `Conv2d` (Cin$\to$Cout), noyau $K_h \times K_w$, sortie $H_{\text{out}} \times W_{\text{out}}$ (stride 1, groups = 1) :

$$\text{MACs} \;=\; H_{\text{out}} W_{\text{out}} \;\times\; \text{Cout} \;\times\; \text{Cin} \;\times\; K_h K_w.$$

Pour une `ConvTranspose2d` (mêmes dimensions de noyau, stride 2 classiquement pour l'upsampling), la formule est identique en remplaçant $H_{\text{out}} W_{\text{out}}$ par l'aire de la *sortie* (niveau plus haut). Toutes nos convolutions 3×3 utiliseront $K_h K_w = 9$; les *down/up* 2×2 utiliseront $K_h K_w = 4$.

Notation à niveaux. DRUNet est un U-Net résiduel avec L niveaux d'encodeur ($\ell = 0, \ldots, L-1$) et un *bottleneck* au niveau L, plus un décodeur symétrique. On note :

$$H_\ell = \frac{H}{2^\ell}, \quad W_\ell = \frac{W}{2^\ell}, \quad r_\ell \stackrel{\text{def}}{=} \frac{H_\ell W_\ell}{HW} = \frac{1}{4^\ell}.$$

Les largeurs (canaux) par niveau sont $n_0, \ldots, n_L$ (croissantes en profondeur). La tête projette $d = C + s$ canaux (image C + carte bruit s) vers n_0. On prend des blocs résiduels composés de **deux** convolutions 3×3.

Somme symbolique (MACs/pixel) – sans fusion séparée. Toutes les quantités ci-dessous sont normalisées par pixel d'entrée via $r_\ell = 4^{-\ell}$:

$$\begin{aligned}\text{MACs/pixel}_{\text{DRUNet}} = \; & \underbrace{9\, r_0\, n_0\, d}_{\text{tête } (C+s)\to n_0} \; + \; \underbrace{\sum_{\ell=0}^{L-1} 18\, r_\ell\, B_\ell^{\text{enc}}\, n_\ell^2}_{\text{blocs résiduels encodeur}} \; + \; \underbrace{\sum_{\ell=0}^{L-1} 4\, r_{\ell+1}\, n_{\ell+1}\, n_\ell}_{\text{down } 2\times 2} \\ & + \; \underbrace{18\, r_L\, B^{\text{bot}}\, n_L^2}_{\text{bottleneck}} \; + \; \underbrace{\sum_{\ell=L-1}^{0} 4\, r_\ell\, n_\ell\, n_{\ell+1}}_{\text{up (transpose) } 2\times 2} \; + \; \underbrace{\sum_{\ell=L-1}^{0} 18\, r_\ell\, B_\ell^{\text{dec}}\, n_\ell^2}_{\text{blocs résiduels décodeur}} \; + \; \underbrace{9\, r_0\, n_0\, C}_{\text{queue } n_0\to C}\,.\end{aligned}$$

Remarque. La "fusion" après concaténation du skip est comptée dans la première 3×3 du bloc du décodeur ; on ne rajoute donc pas de terme séparé.

- B_ℓ^{enc} et B_ℓ^{dec} sont le nombre de blocs résiduels à chaque niveau (2 convs par bloc $\Rightarrow 18\, r_\ell n_\ell^2$).
- Les termes *down/up* sont des conv 2×2 stride 2 ; ils sont comptés à l'aire de leur *sortie* ($r_{\ell+1}$ pour le down, r_ℓ pour l'up).

Ancrage d'implémentation (`deepinv`). Configuration détectée (modèle `deepinv.DRUNet` pré-entraîné) :

$$\begin{aligned}&C = 3, \quad s = 1 \text{ (carte de bruit concaténée)}, \quad d = C + s = 4,\\ &L = 3 \;\; \text{(encodeur à 3 niveaux } \ell = 0, 1, 2, \text{ bottleneck au niveau } L = 3),\\ &[n_0, n_1, n_2, n_3] = [64,\, 128,\, 256,\, 512],\\ &B^{\text{enc}} = [4, 4, 4], \quad B^{\text{bot}} = 4, \quad B^{\text{dec}} = [4, 4, 4],\\ &s_\ell = n_\ell \text{ (concaténation des } \textit{skips} \text{ de même largeur).}\end{aligned}$$

Toutes les convolutions internes utilisent des noyaux 3×3 ($K_h K_w = 9$), et les *down/up* des noyaux 2×2 ($K_h K_w = 4$). On normalise par pixel d'entrée avec $r_\ell = 4^{-\ell}$.

Instanciation numérique (MACs/pixel puis ops/pixel).

$$\begin{aligned}\text{tête} &: 9\, r_0\, n_0\, d = 9 \cdot 1 \cdot 64 \cdot 4 = \mathbf{2\,304},\\ \text{enc. blocs} &: \sum_{\ell=0}^{2} 18\, r_\ell\, B_\ell^{\text{enc}}\, n_\ell^2 = 18\Big(1 \cdot 4 \cdot 64^2 + \tfrac{1}{4} \cdot 4 \cdot 128^2 + \tfrac{1}{16} \cdot 4 \cdot 256^2\Big) = \mathbf{884\,736},\\ \text{down} &: \sum_{\ell=0}^{2} 4\, r_{\ell+1}\, n_{\ell+1}\, n_\ell = 4\Big(\tfrac{1}{4} \cdot 128 \cdot 64 + \tfrac{1}{16} \cdot 256 \cdot 128 + \tfrac{1}{64} \cdot 512 \cdot 256\Big) = \mathbf{24\,576},\\ \text{bottleneck} &: 18\, r_3\, B^{\text{bot}}\, n_3^2 = 18 \cdot \tfrac{1}{64} \cdot 4 \cdot 512^2 = \mathbf{294\,912},\\ \text{up} &: \sum_{\ell=2}^{0} 4\, r_\ell\, n_\ell\, n_{\ell+1} = 4\Big(\tfrac{1}{16} \cdot 256 \cdot 512 + \tfrac{1}{4} \cdot 128 \cdot 256 + 1 \cdot 64 \cdot 128\Big) = \mathbf{98\,304},\\ \text{dec. blocs} &: \sum_{\ell=2}^{0} 18\, r_\ell\, B_\ell^{\text{dec}}\, n_\ell^2 = \mathbf{884\,736},\\ \text{queue} &: 9\, r_0\, n_0\, C = 9 \cdot 64 \cdot 3 = \mathbf{1\,728}.\end{aligned}$$

Somme :

$$\boxed{\text{MACs/pixel}_{\text{DRUNet}} \;=\; \mathbf{2\,191\,296}.} \qquad \boxed{\text{ops/pixel} \approx 2 \times \text{MACs/pixel} = \mathbf{4\,382\,592}.}$$

Table 1: Comparaison des méthodes en termes de PSNR (dB) et de coût opérationnel (ops/pixel).

Méthode	$\sigma = 5$	$\sigma = 10$	$\sigma = 25$	**Coût (ops/pixel)**
Noisy (entrée)	34.25	28.30	20.54	–
TV (Chambolle)	36.25	30.94	27.07	~ 955 ($T = 50$)
BM3D	37.41	33.21	28.26	$\sim 8.5 \times 10^4$
FFDNet (IPOL)	39.59	35.71	30.58	$\sim 4.2 \times 10^5$
DRUNet (deepinv)	40.51	36.35	31.03	$\sim 2.6 \times 10^6$

7 Discussion

Analyse qualitative

Les résultats mettent en évidence une hiérarchie claire entre les méthodes. La TV de Chambolle reste extrêmement légère en coût opérationnel (environ 10^3 opérations par pixel pour $T = 50$), mais ses performances en PSNR demeurent limitées. Elle améliore visiblement les images bruitées, tout en introduisant un aspect trop simplifié et une perte de détails fins.

BM3D offre un compromis intéressant. Avec un coût d'environ 8.5×10^4 opérations par pixel, il permet un gain de 3 à 8 dB selon le niveau de bruit, et produit une qualité perçue nettement supérieure à celle de la TV. C'est pourquoi il reste une référence solide parmi les approches classiques.

FFDNet franchit un cap supplémentaire : il améliore significativement le PSNR (jusqu'à +5 dB à $\sigma = 5$ par rapport à la TV, et près de +4.8 dB à $\sigma = 25$), mais au prix d'un coût environ cinq fois plus élevé que celui de BM3D. Sa force réside dans la flexibilité offerte par la carte de bruit, qui lui permet de traiter différents niveaux σ avec un seul modèle.

Enfin, DRUNet s'impose comme la méthode la plus performante en termes de PSNR (40.5 dB à $\sigma = 5$, 31 dB à $\sigma = 25$). Cette supériorité se paie toutefois cher : son coût atteint environ 2.6×10^6 opérations par pixel, soit près de trente fois celui de BM3D. Son architecture U-Net multi-échelles explique à la fois son efficacité remarquable et sa lourdeur computationnelle.

Validation des comptages analytiques

Afin de vérifier la cohérence des dérivations théoriques, nous avons comparé les résultats analytiques (*MACs/pixel*) aux estimations numériques obtenues avec `ptflops` sur un passage avant des réseaux.

Table 2: Comparaison entre comptage analytique et estimation numérique (ptflops).

Méthode	Analytique (MACs/pixel)	Numérique (MACs/pixel)	Écart abs.	Écart relatif
FFDNet (couleur)	213 192	214 200	1 008	0.47 %
DRUNet	2 191 296	2 119 424	71 872	3.28 %

Analyse.

- Pour **FFDNet**, l'écart est inférieur à 0.5%, ce qui valide pleinement le modèle analytique.
- Pour **DRUNet**, l'écart reste faible ($\sim 3.3\%$), sachant la complexité de l'architecture (skip connections, blocs résiduels, conv. transpose).

- Ces écarts s'expliquent par de légères différences d'implémentation (fusion BN, biais, padding, ou détails de l'upsampling).

Les comptages analytiques sont donc fiables et constituent une base solide pour comparer les méthodes de manière normalisée (*ops/pixel*), tout en étant confirmés par une validation numérique indépendante.

Impact de la nature des données d'entraînement

Il est important de souligner que, contrairement à TV et BM3D qui ne nécessitent pas d'apprentissage préalable, FFDNet et DRUNet doivent leurs performances à un entraînement supervisé sur de grandes bases d'images naturelles (FFDNet sur BSD, DIV2K, Waterloo ; DRUNet sur ImageNet). Ces jeux de données étant proches en nature du dataset de test CBSD68 (images naturelles variées), les deux réseaux généralisent efficacement et obtiennent des PSNR élevés.

Toutefois, cette supériorité pourrait s'atténuer dans d'autres contextes : par exemple en présence de bruit non gaussien (bruit de capteur, artefacts de compression) ou sur des images spécialisées (imagerie médicale, astrophotographie), où les données d'entraînement ne correspondent plus à la distribution test. En revanche, les méthodes classiques comme BM3D ou TV, bien que moins performantes en moyenne, ne souffrent pas de ce biais lié à l'apprentissage et gardent une certaine robustesse hors distribution.

Ainsi, la comparaison doit être nuancée : les méthodes profondes exploitent pleinement la similarité entre leurs données d'entraînement et CBSD68, mais leur efficacité n'est pas garantie de manière universelle.

Références

[1] M. Lebrun, *An Analysis and Implementation of the BM3D Image Denoising Method*, Image Processing On Line, vol. 2, pp. 175–213, 2012. DOI: 10.5201/ipol.2012.l-bm3d.

[2] J. Duran, B. Coll, C. Sbert, *Chambolle's Projection Algorithm for Total Variation Denoising*, Image Processing On Line, vol. 3, pp. 311–331, 2013. DOI: 10.5201/ipol.2013.61.

[3] M. Tassano, J. Delon, T. Veit, *An Analysis and Implementation of the FFDNet Image Denoising Method*, Image Processing On Line, vol. 9, pp. 1–25, 2019. DOI: 10.5201/ipol.2019.231.

[4] K. Zhang, W. Zuo, L. Zhang, *FFDNet: Toward a Fast and Flexible Solution for CNN-based Image Denoising*, IEEE Transactions on Image Processing, vol. 27, no. 9, pp. 4608–4622, 2018. arXiv:1710.04026.

[5] J. Tachella, S. Ehret, G. Peyré, J. Fadili, *DeepInverse: A Python Package for Solving Imaging Inverse Problems with Deep Learning*, arXiv preprint, 2025. arXiv:2505.20160.