

**Optimisation multiobjectif des
hyperparamètres d'un CNN pour orienter le
choix d'un compromis entre exactitude et
efficacité énergétique**

PLAPIED Mathieu

 (Signature pour approbation du dépôt - REE art. 40)

Promoteur : FRENAY Benoît

Mémoire présenté en vue de l'obtention du grade de Master 60 en Sciences Informatiques (HD)

Remerciements

Tout d'abord, je tiens à remercier M. Frenay, promoteur de ce mémoire, pour son accompagnement pédagogique. J'ai eu la chance d'apprendre énormément d'un sujet passionnant.

Ensuite, je tiens à remercier ma mère qui m'a toujours soutenu et accompagné durant mon cursus universitaire.

Enfin, je tiens à remercier mes amis et mes collègues pour avoir été rendu ces années d'études inoubliables.

Résumé

Dans un cadre de transfer learning sur CNN, ce mémoire étudie comment l'optimisation multiobjectif des hyperparamètres avec NSGA-II et Optuna permet de créer un front de Pareto exactitude-énergie et d'identifier un compromis adapté. L'espace de recherche couvre des hyperparamètres clés de l'entraînement comme le learning rate du backbone et du classifier, la weight decay, le dropout, la taille de batch et le nombre de blocs dégelés. L'analyse fANOVA met en évidence que l'énergie d'entraînement dépend surtout du nombre de blocs dégelés, avec près de 93 % de la variance, tandis que l'exactitude est principalement déterminée par le learning rate du backbone, autour de 91 %. Le front de Pareto observé est concave et présente des rendements décroissants. À mesure que l'exactitude augmente, l'efficacité énergétique diminue. Un point de genou atteint 98 % de l'exactitude maximale tout en réduisant d'environ 31 % l'énergie par image. La portée est volontairement limitée à un modèle et un jeu de données, et seule l'énergie d'entraînement est mesurée.

Mots-clés : *CNN, transfer learning, Pareto, fANOVA, efficacité énergétique, optimisation multiobjectif, optimisation des hyperparamètres*

Abstract

This thesis shows, in a transfer-learning setting on CNNs, how multi-objective hyperparameter optimization with NSGA-II via Optuna makes it possible to construct a Pareto front between accuracy and energy per image, and then to derive an operational trade-off. The search space covers key training hyperparameters such as the backbone and classifier learning rates, weight decay, dropout, batch size, and the number of unfrozen blocks. The fANOVA analysis highlights that training energy depends primarily on the number of unfrozen blocks, accounting for about 93 % of the variance, whereas accuracy is mainly determined by the backbone learning rate, at around 91 %. The observed front is concave and exhibits diminishing returns. As accuracy increases, energy efficiency decreases. A knee point attains 98 % of the maximum accuracy while reducing energy per image by about 31 %. The scope is deliberately limited to a single model and dataset, and only training energy is measured.

Keywords : *CNN, transfer learning, Pareto, fANOVA, energy efficiency, multi-objective optimization, hyperparameter optimization*

Table des matières

1	Introduction	1
1.1	Problématique, démarche et objectifs	1
1.2	Questions de recherche	2
1.3	Structure du document	2
2	Fondements : CNN et transfer learning	3
2.1	Bases théoriques	3
2.1.1	Architecture et principes	3
2.1.2	Implications de l'entraînement des CNN depuis zéro	4
2.2	Le transfer learning	6
2.2.1	Principes et définitions	6
2.2.2	Stratégies de transfert learning	7
2.2.3	Comparaison entraînement inférence	10
2.3	Enjeux et limites du transfer learning	11
2.3.1	Domain shift et besoin de fine-tuning	11
2.3.2	Le rôle du modèle dans l'équilibre exactitude-efficacité	13
2.3.3	Évolutions récentes	13
3	Hyperparamètres	16
3.1	Rappel : Distinction entre paramètres et hyperparamètres	16
3.2	L'impact des hyperparamètres sur le compromis exactitude-efficacité	16
3.3	Catégories d'hyperparamètres	18
3.4	Hyperparamètres clés de l'entraînement	18
3.5	Méthode fANOVA	21
4	Métriques : performance et efficacité énergétique	23
4.1	Indicateurs de performance pour les CNN	23
4.2	Indicateurs d'empreinte computationnelle	25
4.3	Indicateurs d'efficacité énergétiques	26
5	Optimisation multiobjectif	29
5.1	Front de Pareto	29
5.1.1	Méthodes de sélection d'un compromis sur le front	30
5.2	Approches évolutionnaires multiobjectif	31
5.2.1	Principes des algorithmes évolutionnaires	31

5.2.2	NSGA-II	32
5.2.3	Algorithmes évolutionnaires multiobjectifs alternatifs	35
5.3	Optimisation multiobjectif avec Optuna	36
6	Méthodologie expérimentale	38
6.1	Environnement matériel et logiciel	38
6.2	Modèle, jeu de données, optimiseur et métriques	38
6.3	Optimisation des hyperparamètres	42
6.3.1	Espace de recherche : hyperparamètres	43
6.3.2	Configuration NSGA-II	44
6.3.3	Importance des hyperparamètres par objectif	44
6.4	Entrainement haute fidélité	44
7	Résultats : analyse et discussion	46
7.1	Front de Pareto	46
7.2	Entrainement haute fidélité	48
7.3	Importance des hyperparamètres	48
8	Conclusion	51

Table des figures

1	Exemple de sous-échantillonnage	4
2	Architecture d'un CNN	4
3	Illustration du transfer learning	8
4	Illustration du front de Pareto	30
5	Diagramme illustratif du fonctionnement d'un algorithme évolutionnaire.	33
6	Illustration de deux aspects de NSGA-II	34
7	Échantillon d'images présentes dans le jeu de données Stanford Dogs.	41
8	Front de Pareto obtenu après l'optimisation multiobjectif	46
9	Les trois configurations choisies sur le front de Pareto.	47
10	Importance des hyperparamètres par rapport à l'efficacité en %.	49
11	Importance des hyperparamètres par rapport à l'exactitude en %.	49

Acronyms

FC Fully Connected.

RNN Recurrent Neural Network.

TL Transfer Learning.

DTL Deep Transfer Learning.

SOTA State of the Art.

FLOPs Floating-Point Operations.

GPU Graphics Processing Unit.

CPU Central Processing Unit.

TPU Tensor Processing Unit.

NPU Neural Processing Unit.

FPGA Field-Programmable Gate Array.

FLOP Floating-point Operation.

1 Introduction

L'émergence du deep learning, et plus particulièrement des réseaux de neurones convolutifs (CNN), a profondément transformé de nombreux domaines industriels. Porté par un large éventail d'applications tel que la reconnaissance faciale, la classification d'images, la reconnaissance de texte ou encore l'apprentissage par renforcement, cet essor s'est accompagné d'une hausse significative des coûts computationnels et énergétiques. Dans les contextes industriels comme académiques, l'enjeu n'est plus seulement d'atteindre la meilleure exactitude possible, mais aussi d'y parvenir sous des contraintes de ressources, de temps ou d'énergie. Cette contrainte est renforcée par le fait que la conception d'un système implique souvent plusieurs cycles d'ajustement de l'architecture, des hyperparamètres et des entraînements. Dans ce cadre, le transfer learning apparaît comme une solution efficace pour réduire le coût d'entraînement en réutilisant des représentations apprises sur des jeux de données volumineux et variés. Toutefois, même dans un contexte de TL, la performance et l'empreinte énergétique restent fortement influencées par les hyperparamètres, dont les effets sont souvent interdépendants et non linéaires.

Chercher un compromis entre exactitude et efficacité énergétique revient à formuler un problème multiobjectif. Au lieu de viser une solution unique, il s'agit de caractériser un ensemble de solutions non dominées, où chacune d'elles offre un équilibre différent entre les objectifs. Les solutions non dominées signifient qu'il n'est pas possible d'améliorer un objectif sans en dégrader un autre (front de Pareto). Ce principe est particulièrement utile lorsque les contraintes varient d'un contexte à l'autre. Un même projet peut ainsi faire des choix différents selon que la priorité soit donnée à un solution moins énergivore, à la rapidité d'itération ou à la meilleure exactitude atteignable sans contrainte.

Dans la pratique, ajuster manuellement des hyperparamètres, surtout lorsqu'ils sont nombreux, est chronophage et peu robuste, d'où l'émergence d'outils d'optimisation automatisés tels qu'Optuna ou SMAC. Les interactions entre ces variables font qu'une amélioration locale peut dégrader le comportement global. Par exemple, un *learning rate* trop élevé risque de nuire à la convergence du modèle, tandis qu'une valeur trop faible peut ralentir l'entraînement et bloquer le modèle dans une solution sous-optimale. Dégeler davantage de couches facilite l'adaptation au domaine cible, mais accroît le coût par image (J/image). D'où l'intérêt d'une optimisation automatisée des hyperparamètres, qui explore systématiquement l'espace de recherche et produit un front de Pareto plutôt qu'un réglage unique.

1.1 Problématique, démarche et objectifs

L'essor des CNN s'accompagne d'un coût d'entraînement élevé en temps et en énergie. Même en transfer learning, les performances et l'empreinte énergétique dépendent fortement des hyperparamètres dont les effets sont interdépendants. Il est donc pertinent de formuler la recherche d'un compromis exactitude-efficacité comme un problème multiobjectif produisant un front de Pareto plutôt qu'un réglage unique.

Ce mémoire documente par un état de l'art et illustre, à travers un cas d'étude, l'usage de l'optimisation multiobjectif des hyperparamètres pour allier exactitude et efficacité énergétique en TL sur CNN. Pour une tâche de classification d'images, un espace de recherche centré sur quelques hyperparamètres clés de l'entraînement est défini (*dropout*, *batch size*, *weight decay*, *learning rate*, nombre de couches dégelées). Une optimisation multiobjectif génère des configurations qui maximisent l'exactitude sur un jeu de validation tout en minimisant un indicateur

énergétique d’entraînement (J/image). Le front obtenu est analysé et interprété en fonction de l’importance relative des hyperparamètres afin d’identifier des leviers de décision. L’objectif n’est pas de proposer une nouvelle méthode d’optimisation ni de battre des *benchmarks*, mais de montrer concrètement comment construire et exploiter le front pour choisir un compromis.

Le périmètre de ce travail est restreint aux points suivants : (i) il porte exclusivement sur des CNN en TL, (ii) l’optimisation est illustrée sur un seul couple modèle/jeu de données, sans chercher à comparer toutes les méthodes, (iii) seule l’énergie d’entraînement est mesurée, l’impact de l’inférence sur la consommation d’énergie est discuté mais non évalué expérimentalement, (iv) les résultats fournissent uniquement des repères pour la prise de décision dans ce contexte précis, sans valoir de règle générale.

1.2 Questions de recherche

Deux questions guident le mémoire :

1. Dans un cadre de transfer learning sur des CNN, comment utiliser une optimisation multiobjectif pour construire un front de Pareto entre exactitude et efficacité énergétique et comment ce front peut-il guider le choix d’un compromis adapté à un cas d’étude ?
2. Quels hyperparamètres influencent le plus la position des solutions sur le front et comment les identifier de façon fiable pour aider la prise de décision ?

1.3 Structure du document

Ce travail est organisé de la façon suivante : La section 2 présente les fondements des CNN, tels que leurs principes, leur architecture et la provenance des couts d’entraînement. Cette section présente aussi les notions et stratégies de transfer learning, les enjeux, dont le *domain shift*. La section 3 traite des hyperparamètres, la distinction avec les paramètres et leur accent sur quelques leviers centraux, ainsi que la méthode fANOVA pour estimer leur importance. La section 4 synthétise les métriques de performance prédictive utilisées par les CNN ainsi que les indicateurs computationnels et énergétiques pertinents à l’entraînement. La section 5 formalise l’optimisation multiobjectif en définissant le front de Pareto, la sélection d’un compromis et une brève mise en perspective des approches évolutionnaires utilisées en pratique. La section 6 décrit la méthodologie expérimentale, comprenant l’environnement, le modèle et les données, ainsi que l’espace d’hyperparamètres et le protocole d’optimisation. La section 7 présente les résultats obtenus et leur discussion. Enfin, la section 8 conclut et ouvre sur des pistes d’amélioration.

2 Fondements : CNN et transfer learning

Cette section expose les bases théoriques nécessaires à la compréhension de ce travail. Elle commence par une présentation des principes et de l'architecture des réseaux de neurones convolutifs (CNN), ainsi que des implications de leur entraînement en termes de coût computationnel et énergétique. Elle introduit ensuite la notion de transfer learning, en expliquant ses principes et ses différentes stratégies de mise en oeuvre. Les limites liées au phénomène de domain shift seront évoquées. Enfin, cette section examine le rôle du modèle source dans l'équilibre entre exactitude et empreinte énergétique, ainsi que certaines évolutions récentes, telles que la distillation et la compression des modèles, qui participent à améliorer l'efficacité des CNN dans un contexte d'optimisation multiobjectifs.

2.1 Bases théoriques

2.1.1 Architecture et principes

Les CNN (Convolutional Neural Networks) constituent un paradigme majeur et une catégorie importante de modèles en deep learning. Ils sont largement utilisés dans des tâches variées telles que la classification d'images, la détection d'objets ([Li et al.](#)), la reconnaissance de texte ([Wang et al.](#)), la reconnaissance faciale ([Hu et al.](#)) ou encore l'apprentissage par renforcement ([Baker et al.](#)).

Les CNN sont des réseaux de neurones à propagation avant (*feedforward*) qui se distinguent par l'utilisation de structures convolutionnelles permettant d'extraire automatiquement des caractéristiques pertinentes à partir des données. Cette capacité d'extraction automatique constitue l'aspect fondamental de cette technologie.

L'architecture d'un CNN se divise généralement en deux parties essentielles : un module d'extraction de caractéristiques et un classificateur. Le premier est chargé d'extraire des caractéristiques à partir des données brutes, tandis que le second prend ces représentations comme entrée pour effectuer une prédiction. Cette architecture repose sur plusieurs types de couches, telles que les couches de convolution, les couches de subdivision (*pooling*), les couches complètement interconnectées (*fully-connected*), combinées à des fonctions d'activation non linéaires. Fig 2 représente l'architecture d'un CNN et comment les données le traversent. L'entrée représente généralement une image RGB, où chaque couleur (rouge, vert, bleu) correspond à un canal distinct.

Couche de convolution La couche de convolution est la partie centrale d'un CNN. C'est elle qui permet d'extraire les caractéristiques de l'entrée. Il y a plusieurs couches de convolution : les premières permettent d'extraire des caractéristiques plus simples, telles que des lignes, des bords ou des textures, tandis que les couches plus proches de la sortie du réseau capturent des caractéristiques plus abstraites. Pour extraire ces caractéristiques, un kernel (ou filtre), qui est une matrice de poids (par exemple de taille 3×3), se déplace sur l'entrée pour former une sortie appelée carte de caractéristiques qui sera à son tour utilisée par la couche de convolution suivante.

Couche de sous-échantillonnage (*pooling layer*) La couche de sous-échantillonnage permet de réduire la taille des cartes de caractéristiques et le nombre de connexions dans le modèle, en effectuant un *downsampling* et une réduction de la dimensionnalité des données en entrée. Il

existe plusieurs méthodes de sous-échantillonnage, telles que le *MaxPooling* et l'*AveragePooling*. D'autres variantes peuvent également être utilisées selon les besoins spécifiques du modèle. Le MaxPooling applique un filtre de taille définie qui se déplace sur la sortie de la couche de convolution précédente et conserve uniquement la valeur maximale dans cette fenêtre (fig 1). Cette technique permet de réduire considérablement la taille des cartes de caractéristiques et de rendre le modèle plus robuste face au surapprentissage.

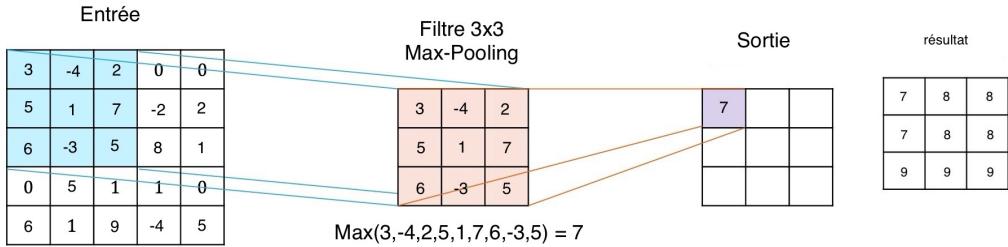


FIGURE 1: Exemple de sous-échantillonnage qui préserve uniquement les caractéristiques les plus importantes ([Wikipedia contributors](#)).

Couche entièrement connectée (FC layer) : La couche entièrement connectée se trouve à la fin de l'architecture, juste après une couche de sous-échantillonnage ou de convolution. Dans celle-ci, chaque neurone est connecté à tous les neurones de la couche précédente, comme c'est également le cas dans les réseaux de neurones récurrents (RNN). La carte des caractéristiques est alors aplatie en un vecteur unidimensionnel afin d'être compatible avec cette couche. La couche entièrement connectée est utilisée pour permettre au réseau de réaliser des tâches de classification ou de prédiction ([Shiri et al.](#)).

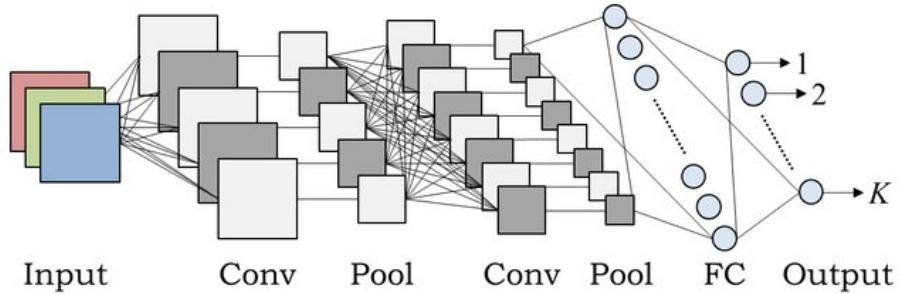


FIGURE 2: Architecture d'un CNN ([Hidaka et Kurita](#)).

2.1.2 Implications de l'entraînement des CNN depuis zéro

L'entraînement d'un réseau de neurones convolutifs se déroule en plusieurs phases. La première est la propagation vers l'avant (*forward pass*). Les images d'entrée traversent successivement les couches convolutionnelles, qui extraient des caractéristiques locales à partir de régions spécifiques de l'image. Ensuite, les couches de *pooling* conservent uniquement les caractéristiques les plus significatives. À la fin du parcours, les couches entièrement connectées (*fully connected*) aplatisent les représentations extraites en un vecteur unidimensionnel, permettant ainsi de produire une sortie finale, généralement une probabilité de classification dans le cadre d'une tâche supervisée. Cette suite d'opérations constitue la passe avant, au cours de laquelle aucun paramètre du réseau n'est modifié. C'est la fonction de perte (*loss function*) qui permet de quantifier l'écart entre la sortie du réseau et le résultat attendu. Cet écart est ensuite utilisé lors de la propagation vers

l’arrière (*backpropagation*) pour mettre à jour les paramètres du réseau dans le but de minimiser l’erreur globale. La backpropagation est appliquée à chaque *batch* de données, qui représente un sous-ensemble du jeu de données, typiquement composé de 32, 64 ou 128 images. Une fois que tous les lots ont été traités, on considère qu’une *epoch* d’entraînement est terminée. À la fin de chaque epoch, le modèle est évalué sur un jeu de test afin de mesurer l’évolution de la fonction de perte et la précision du modèle. Ce processus est répété sur plusieurs epochs jusqu’à l’obtention de performances satisfaisantes.

Les hyperparamètres jouent un rôle crucial pendant la phase d’entraînement, car ils influencent directement les performances du modèle. Définis avant l’entraînement, ils encadrent le processus d’apprentissage. Contrairement aux paramètres internes du modèle, les hyperparamètres ne sont pas appris automatiquement. Ils sont fixés manuellement ou optimisés par des méthodes spécifiques. L’espace des hyperparamètres étant vaste, leur réglage constitue une étape complexe et chronophage. Pour mieux comprendre leur impact, certaines méthodes comme fANOVA (functional ANOVA), introduites par [Hutter et al.](#), permettent d’estimer l’importance relative de chaque hyperparamètre en analysant leur influence sur la performance du modèle. Cette approche fournit un classement des hyperparamètres les plus influents.

Voici quelques exemples d’hyperparamètres qui influencent directement le temps d’entraînement, et donc la consommation d’énergie :

- **Learning rate** : détermine la vitesse à laquelle les poids sont mis à jour. Un taux trop élevé peut empêcher la convergence, tandis qu’un taux trop faible ralentit l’apprentissage.
- **Batch size** : taille des lots de données utilisés à chaque itération. Une taille plus grande accélère l’entraînement, mais nécessite plus de mémoire.
- **Nombre d’epochs** : nombre de fois que l’ensemble du jeu de données est parcouru. Trop d’epochs peuvent entraîner un surapprentissage.
- **Dropout** : taux de désactivation aléatoire de neurones pendant l’entraînement pour réduire le surapprentissage.

Il existe d’autres hyperparamètres importants mais qui ne sont généralement pas modifiables dans le cadre du transfer learning, car ils touchent à l’architecture du réseau. Par exemple : la taille et le nombre de filtres par couche, le *padding* qui ajoute des pixels autour de l’image pour préserver les dimensions, la *stride* qui contrôle le décalage de la fenêtre de convolution (une stride plus grande réduit la taille de la sortie et accélère l’entraînement au prix d’une perte de précision), ou encore la fonction d’activation, qui détermine l’activation des neurones.

Compte tenu de la multitude d’hyperparamètres à ajuster, du nombre d’itérations nécessaires, et de la complexité des architectures utilisées, entraîner un modèle depuis zéro devient rapidement très coûteux en ressources. Cela s’explique par plusieurs facteurs :

- **Multiplication matricielle intensive** : chaque couche de convolution implique un grand nombre de calculs matriciels. Les vecteurs en entrée sont multipliés par des matrices (filtres) selon une fenêtre glissante, générant une sortie à chaque position du filtre ([Du-moulin et Visin](#)). En raison du nombre de filtres et de la taille des entrées, le nombre total d’opérations par couche peut devenir très élevé.
- **Architectures volumineuses** : la charge computationnelle dépend fortement de la profondeur du réseau, de sa largeur et de la taille des images. Selon [Tan et Le](#), doubler la largeur du réseau ou la résolution des images multiplie par quatre le nombre total d’opérations (FLOPs¹), rendant les modèles très gourmands en ressources.

1. Bien que FLOPS désigne généralement une unité de performance (floating-point operations per second), nous utilisons ici FLOPs pour désigner le nombre total d’opérations nécessaires à l’entraînement.

- **Rétropropagation** : ce processus d'apprentissage consiste à calculer les gradients de l'erreur pour ajuster les poids du réseau. Il implique des opérations complexes sur l'ensemble des couches, avec une mise à jour de millions de paramètres à chaque itération.

Pour replacer cette théorie dans un contexte réel, l'entraînement d'un modèle comme ResNet-50 sur le dataset MNIST (redimensionné en 48×48) consomme environ 3 kWh [Xu et al.](#), soit l'équivalent d'un réfrigérateur fonctionnant pendant 4,3 jours². MNIST reste toutefois un jeu de données simple, adapté à des tâches élémentaires. La plupart des cas d'usage des CNN impliquent des jeux de données bien plus volumineux, comme ImageNet, nécessitant davantage de ressources computationnelles. De plus, les tâches associées sont souvent plus complexes, ce qui rallonge le temps de convergence.

Cette approche d'entraînement depuis zéro, en plus d'être coûteuse en ressources et en énergie, peut ne pas toujours aboutir aux performances espérées. C'est dans ce contexte que le transfer learning s'impose comme une alternative efficace, en permettant de réutiliser un modèle pré-entraîné.

Cependant, il n'est pas toujours nécessaire d'entraîner un modèle depuis zéro pour une nouvelle tâche. Grâce au transfer learning, il est possible de réutiliser un modèle préalablement entraîné sur une première tâche pour l'adapter à une seconde. Cette technique permet d'éviter de réentraîner un modèle depuis le début. Les poids des neurones appris lors de la première tâche sont réutilisés, au lieu d'initialiser aléatoirement de nouveaux poids [Gupta et al.](#). Cela permet de réduire le temps de calcul et de rendre possible l'apprentissage à partir de jeux de données restreints.

2.2 Le transfer learning

2.2.1 Principes et définitions

Le transfer learning est une méthode d'apprentissage qui vise à utiliser les connaissances acquises d'un domaine source pour faciliter l'apprentissage d'une nouvelle tâche cible. Cette approche est inspirée de la psychologie où il est possible de généraliser une expérience pour la transférer d'une situation à une autre. Par exemple, apprendre à faire du vélo pourra être utile pour apprendre à faire de la moto. Le machine learning a connu un développement important ces dernières années et a réussi à s'appliquer pour résoudre un grand nombre de problèmes pratiques. Idéalement, pour le bon fonctionnement d'un algorithme de machine learning, il faut beaucoup de données étiquetées qui ont la même distribution que le jeu de test. En pratique, il est souvent coûteux, long, voire irréalisable d'obtenir des données étiquetées, comme par exemple dans le milieu médical où les données sont souvent en quantité très limitée ([Swati et al.](#)). Le machine learning classique montre ses limites lorsqu'il n'existe pas suffisamment de données étiquetées. Une solution partielle qui limite cette dépendance aux données étiquetées est d'utiliser des approches semi-supervisées, qui combinent un petit nombre de données étiquetées avec un grand nombre de données non étiquetées. Cependant, même la collecte de données non étiquetées peut s'avérer difficile, et ces approches n'aboutissent pas toujours à des performances satisfaisantes. Le TL minimise le nombre d'exemples étiquetés nécessaires à l'entraînement pour apprendre une nouvelle tâche cible ; de manière générale, la distribution des données source et cible est différente.

Il faut noter que le TL présente tout de même certaines limites. S'il y a peu de similarité entre le domaine source et cible (inter-domaine), le transfer learning risque d'être inefficace. Il est

2. Selon un réfrigérateur domestique moyen consommant 0,7 kWh par jour.

aussi possible que les similarités entre le domaine source et la tâche cible aient un effet négatif sur l'apprentissage, ce problème porte le nom de *negative transfer learning*. Globalement, le transfer learning dépend de plusieurs facteurs : la pertinence entre le domaine source et la tâche cible ; la capacité du modèle à correctement trouver les connaissances transférables intéressantes et utiles à l'apprentissage.

Cette base théorique permet d'introduire le *deep transfer learning* (DTL), qui conserve le même objectif que le transfert learning classique. Réutiliser des connaissances acquises sur une tâche source pour améliorer l'apprentissage sur une tâche cible, mais dans un cadre spécifique aux réseaux de neurones profonds, en particulier les CNN.

Contrairement aux méthodes de machine learning classiques, les modèles profonds nécessitent d'importants volumes de données annotées et des ressources de calcul élevées pour être entraînés efficacement. Le DTL répond à ces contraintes en adoptant une approche centrée sur le modèle (parameter-based). Les poids qui ont été optimisés pour répondre à une tâche source sont réajustés pour en tirer parti dans une nouvelle tâche. Cette approche, qui est spécifique aux architectures profondes, permet non seulement de réduire le coût computationnel mais aussi de limiter les besoins en données annotées pour la tâche cible. Contrairement aux modèles de machine learning qui sont moins dépendants de la taille du jeu de données et dont l'entraînement est généralement moins coûteux car ils sont plus adaptés à des problèmes linéaires. Le DTL peut utiliser les connaissances d'une autre tâche même si elles sont peu compatibles, ce qui réduit le temps d'entraînement ([Iman et al.](#)).

Le DTL est principalement *modèle-centré* : il consiste à manipuler directement les couches d'un réseau pré-entraîné. Trois grandes stratégies dominent :

- **Le *fine-tuning complet***, qui ajuste tout ou partie des couches du modèle en réutilisant les poids préalablement appris ;
- **Le *freezing des couches convolutionnelles***, où seules les dernières couches (souvent pleinement connectées) sont réentraînées, les premières couches étant gelées pour conserver l'extraction de caractéristiques génériques ;
- **L'apprentissage progressif (progressive learning)**, où l'on gèle entièrement le modèle pré-entraîné et l'on ajoute de nouvelles couches entraînées spécifiquement sur la tâche cible.

Chacune de ces méthodes vise à équilibrer la réutilisation efficace des connaissances avec la capacité d'adaptation au nouveau domaine. Par exemple, le freezing limite le risque d'oubli catastrophique, mais peut rendre le modèle trop rigide si le domaine cible diffère fortement du domaine source. À l'inverse, le fine-tuning complet permet une adaptation fine, mais au risque d'effacer les connaissances acquises précédemment. L'approche progressive cherche un compromis, en ajoutant des couches sans modifier le cœur du modèle ([Iman et al.](#)).

2.2.2 Stratégies de transfert learning

À partir des principes précédemment exposés, plusieurs stratégies concrètes d'entraînement peuvent être mises en oeuvre dans le cadre du DTL. Celles-ci consistent à réutiliser, en tout ou en partie, un modèle déjà pré-entraîné sur une tâche source, puis à l'adapter à une tâche cible via une nouvelle phase d'apprentissage, souvent appelée *fine-tuning*.

Dans ce contexte, les poids du modèle sont initialisés à partir de ceux appris lors de la tâche source, et partiellement ou totalement mis à jour selon la similarité entre les jeux de données et les objectifs des tâches concernées. Comme le soulignent [Plested et Gedeon](#), le jeu de données

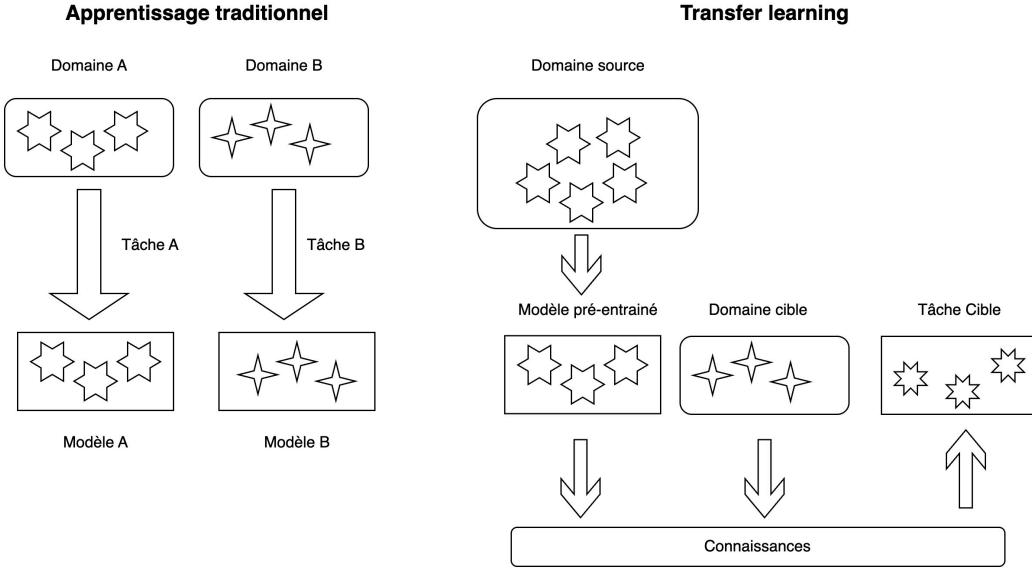


FIGURE 3: Illustration du transfer learning inspiré de ([Gao et al.](#))

source est généralement plus volumineux et diversifié que celui de la tâche cible. Cette différence de taille justifie le recours au transfer, en particulier lorsque les données annotées sont rares ou coûteuses à produire.

Plus les domaines source et cible sont proches, plus il est pertinent de transférer un grand nombre de couches du réseau. À l'inverse, si les tâches sont très différentes, il peut être nécessaire de ne réutiliser que les premières couches du réseau, celles chargées de l'extraction de caractéristiques générales, tout en réentraînant les couches finales spécifiques à la nouvelle tâche.

Extraction de caractéristiques Cette stratégie consiste à conserver uniquement la partie du réseau préentraîné chargée d'extraire les caractéristiques de l'entrée, comme les couches de convolution ou de pooling. Les couches entièrement connectées sont supprimées et remplacées par de nouvelles couches adaptées à la tâche cible. Lors de l'entraînement, les couches conservées sont gelées (non mises à jour), et seules les nouvelles couches sont ajustées. Cette approche est particulièrement pertinente lorsque les tâches source et cible, ainsi que leurs jeux de données, présentent une forte similarité.

Fine-tuning complet Cette approche consiste à mettre à jour l'ensemble des poids d'un modèle préentraîné lors de son adaptation à une tâche cible. Bien qu'elle offre généralement de bonnes performances, elle présente plusieurs limites. Tel qu'une consommation importante de ressources (temps d'entraînement, mémoire, stockage), un risque plus grand de surapprentissage sur les petits jeux de données, et une réutilisabilité réduite du modèle pour d'autres tâches ([Lian et al.](#)). Il est également reconnu que cette approche donne de meilleurs résultats lorsque le jeu de données cible est suffisamment large pour éviter le surapprentissage.

Fine-tuning partiel Le fine-tuning partiel consiste à réentraîner uniquement certaines couches du modèle pré-entraîné, en gelant les autres. Cette stratégie est particulièrement adaptée lorsque la similarité entre les tâches source et cible est intermédiaire : suffisante pour réutiliser une partie des représentations apprises, mais nécessitant tout de même une certaine adaptation.

Concrètement, seules les couches proches de la sortie sont généralement dégelées, car elles sont les plus sensibles aux spécificités de la tâche. Le choix du point de découpe entre les couches figées et celles ajustées dépend de plusieurs facteurs, comme la quantité de données disponibles, la complexité du problème cible ou encore les contraintes computationnelles.

Fine-tuning progressif Le fine-tuning progressif (*layer-wise*) est une approche hybride qui consiste à affiner progressivement les poids d'un modèle en dégelant ses couches une à une, en partant des plus profondes. L'architecture est généralement découpée en blocs, souvent délimités par les couches de pooling, pour réduire le nombre d'itérations nécessaires à l'entraînement. À chaque étape, un bloc supplémentaire est dégelé, et le modèle est réentraîné dans cette nouvelle configuration. Chaque itération est indépendante, et le processus s'arrête lorsque les performances atteignent le niveau souhaité. Les couches proches de la sortie sont dégelées en premier, car elles capturent des caractéristiques plus spécifiques à la tâche. Cette méthode donne de bons résultats, en particulier lorsque les données disponibles sont peu nombreuses mais similaires à celles du pré-entraînement. Par exemple, [Swati et al.](#) ont utilisé cette approche pour la détection de tumeurs cérébrales sur des IRM, en divisant VGG19 en six blocs définis par les couches de pooling. Cela a permis de limiter le nombre d'itérations et d'atteindre une précision globale de 94,82 %.

Les différentes stratégies de transfert learning se distinguent principalement par la proportion de poids du modèle pré-entraîné qui sont modifiés lors de l'entraînement. L'extraction de caractéristiques se limite à l'utilisation fixe des couches de base, tandis que le fine-tuning complet actualise l'ensemble du réseau. Entre ces deux extrêmes, le fine-tuning partiel et progressif permettent d'ajuster le modèle de manière plus ciblée, en fonction du degré de similarité entre les tâches source et cible, des contraintes de calcul, ou encore de la taille du jeu de données disponible. Le choix de la stratégie la plus adaptée repose donc sur un compromis entre performance, coût computationnel et capacité de généralisation.

TABLE 1: Résumé des principales stratégies de (deep) transfer learning

Type	Fonctionnement
Extraction de caractéristiques	Conserve uniquement les couches d'extraction (convolutions, pooling) du modèle pré-entraîné, gelées pendant l'apprentissage. Les couches finales sont remplacées et entraînées pour la tâche cible.
Fine-tuning complet	Met à jour tous les poids du modèle pré-entraîné pour la tâche cible. Offre souvent de bonnes performances mais demande plus de ressources et comporte un risque de surapprentissage sur de petits jeux de données.
Fine-tuning partiel	Ne met à jour qu'une partie des couches (souvent proches de la sortie) en gelant les autres. Compromis idéal lorsqu'il y a une similarité entre les tâches, permettant d'adapter le modèle tout en préservant certaines représentations.
Fine-tuning progressif	Dégèle et réentraîne les couches par étapes, en partant des plus proches de la sortie, puis en remontant progressivement. Limite le nombre d'itérations et s'adapte bien aux petits jeux de données similaires à la tâche source.

Parmi ces approches, certaines seront utilisées dans le cadre du protocole expérimental

présenté au chapitre 6, afin d'évaluer leur influence sur le compromis entre exactitude et efficacité énergétique.

2.2.3 Comparaison entraînement inférence

Il est important de distinguer clairement les phases d'entraînement et d'inférence dans le cycle de vie d'un modèle de deep learning. L'inférence repose uniquement sur une forward pass, c'est-à-dire le passage d'une entrée (par exemple une image) à travers les couches successives du réseau pour produire une prédiction. Contrairement à l'entraînement, elle ne déclenche ni rétropropagation ni mise à jour des poids. Cette opération est donc beaucoup moins intensive en calcul, car elle ne nécessite que des multiplications matricielles sans calcul de gradients. En pratique, l'inférence peut être jusqu'à 50 fois moins coûteuse que l'entraînement, selon les modèles et la configuration utilisée. Toutefois, dans des contextes de déploiement intensif, comme en cloud ou sur des dispositifs embarqués, elle peut représenter jusqu'à 90 % de l'énergie totale consommée sur l'ensemble du cycle de vie du modèle ([Desislavov et al.](#)). Si chaque inférence est peu coûteuse individuellement, leur fréquence d'exécution massive peut en faire une dépense énergétique très importante. Un exemple illustratif est fourni par le modèle EfficientNet-B1. Il surpasse ResNet-152 en précision top-1 sur ImageNet (78,8 % contre 77,8 %) tout en nécessitant 16 fois moins de calculs (FLOPs), 7,69 fois moins de paramètres et une latence d'inférence 5,7 fois plus faible ([Tan et Le](#)). Cet exemple montre que l'optimisation de l'inférence est un levier majeur pour réduire la consommation énergétique des modèles CNN. Par ailleurs, la consommation énergétique ne dépend pas uniquement du GPU. Selon l'article *Carbontracker*, la mémoire vive et le CPU peuvent représenter jusqu'à 30 % de l'énergie utilisée durant l'entraînement ([Anthony et al.](#)). La forward pass, en tant qu'opération plus simple, réduit la pression sur ces composants, car elle ne nécessite pas le stockage temporaire des activations intermédiaires ou des mini-batchs. Enfin, il est essentiel de noter que l'efficacité énergétique de l'inférence dépend fortement du matériel utilisé, car un même modèle peut avoir une empreinte énergétique très différente selon qu'il est exécuté sur un smartphone, un serveur GPU ou une puce FPGA embarquée.

Une méthode pour comparer l'efficacité énergétique d'un modèle est d'utiliser comme métrique le nombre de FLOPS par Watt et FLOPs par inférence. Ces métriques permettent d'évaluer le compromis entre performance de calcul et efficacité énergétique, indépendamment du matériel. C'est ce qu'ont utilisé [Desislavov et al.](#) dans leur étude qui démontre que la tendance de consommation d'énergie n'est pas à une montée anormale pour les CNN malgré des modèles toujours plus complexes. Les modèles de pointe (SOTA) sont très souvent énergivores à l'inférence, mais sont aussi suivis par des alternatives bien plus efficaces. Ce sont ces modèles qui sont utilisés dans des applications industrielles.

On peut aussi noter que, malgré l'augmentation du nombre de paramètres, l'énergie requise n'augmente pas de manière incontrôlée, car des améliorations matérielles et algorithmiques accompagnent cette croissance. Par exemple, EfficientNet-B3 (2019) dispose d'un nombre de paramètres similaire à DenseNet-169 (2017) : 12M contre 14M respectivement, mais parvient à une précision plus élevée (81,1 % contre 76,2 %) tout en nécessitant deux fois moins de calculs, 1,8G contre 3,5G. Cela montre que des gains en efficacité peuvent être obtenus sans nécessairement réduire la taille du modèle, grâce à une meilleure conception architecturale et à des stratégies de *scaling* plus efficaces ([Tan et Le](#)). Ce type d'optimisation algorithmique peut être renforcé lorsqu'il est combiné à des améliorations matérielles ciblées, comme l'usage de processeurs spécialisés tels que les TPU, NPU ou de techniques telles que la quantification, qui permettent d'améliorer significativement la efficacité énergétique à l'inférence.

L'utilisation du transfer learning permet de réduire significativement le nombre d'opérations nécessaires et le temps requis lors de l'entraînement ([Gupta et al.](#)). Cet avantage ne se traduit pas directement par un gain énergétique lors de l'inférence. En effet, l'inférence dépend principalement de l'architecture du modèle, et non de la méthode d'entraînement. Toutefois, dans les cas où le modèle n'est pas destiné à être déployé massivement ou si l'entraînement est répété fréquemment, l'économie réalisée lors de l'entraînement reste un bénéfice énergétique non négligeable. Le TL joue également un rôle clé dans la réduction de l'empreinte carbone à plus large échelle. En facilitant l'utilisation de modèles pré-entraînés compacts, comme MobileNet ou EfficientNet, il permet de limiter la consommation énergétique à l'inférence, en particulier sur des dispositifs contraints (smartphone, domotique). De plus, il favorise le partage des ressources. Un même modèle peut ainsi être affiné pour plusieurs tâches spécifiques, ce qui limite le besoin de réentraîner des architectures lourdes à chaque nouveau cas d'usage.

Cette capacité à réutiliser efficacement les connaissances s'inscrit dans une logique de Green AI, qui vise à limiter l'impact environnemental de l'IA sans compromettre la performance. Ainsi, le transfer learning peut être considéré comme un levier stratégique, permettant de combiner efficacité énergétique et réutilisabilité pendant toute la durée de vie des modèles.

2.3 Enjeux et limites du transfer learning

2.3.1 Domain shift et besoin de fine-tuning

Dans un contexte de TL, on suppose idéalement que la distribution des données du domaine cible est similaire à celle du domaine source ayant servi à l'entraînement initial du modèle. En pratique, cette supposition n'est souvent pas respectée. La distribution des données peut changer entre le domaine source et le domaine cible, un phénomène connu sous le nom de *domain shift*, c'est-à-dire un décalage de distribution entre les deux domaines. L'étude de [Liu et al.](#) a montré que l'erreur mesurer sur le jeu de test d'un modèle tend à augmenter proportionnellement à la divergence entre les distributions d'entraînement et de test . En d'autres termes, plus le domain shift est important, plus les performances du modèle entraîné sur le domaine source risquent de se dégrader sur le domaine cible. Ce problème de généralisation hors distribution est bien documenté pour les CNN ([Chen et al.](#); [Yan et al.](#)). Par exemple, un réseau de classification entraîné sur ImageNet voit son exactitude chuter de manière significative lorsqu'il est évalué sur des images issues d'un domaine légèrement différent et sans adaptation ([Koh et al.](#)). Il est donc important de prendre en compte ce changement de distribution lors du transfert de modèles. En particulier dans une optique d'optimisation de l'empreinte énergétique où l'on cherche à réutiliser des modèles existants plutôt que de tout réentraîner depuis zéro.

Le fine-tuning du modèle est la stratégie généralement adoptée pour faire face à un domain shift entre le domaine source et le domaine cible. Le fine-tuning consiste à réentraîner un modèle préentraîné sur la nouvelle tâche cible, en ajustant tout ou partie de ses poids à l'aide de données annotées du domaine cible comme décrit dans la section [2.2.2](#). Sans fine-tuning, un modèle préentraîné appliqué tel quel à un domaine éloigné risque de ne pas activer les bonnes caractéristiques ou de produire des prédictions biaisées, d'où l'intérêt d'ajuster ses paramètres sur des exemples du nouveau domaine. D'un point de vue pratique, le fine-tuning requiert typiquement moins de données et de temps de calcul que d'entraîner un modèle depuis une initialisation aléatoire, ce qui réduit d'autant la consommation énergétique ([Zhuang et al.](#)). Le fine-tuning apparaît donc comme une solution simple et efficace pour diminuer les effets négatifs du domain shift.

Selon la nature de ce décalage entre domaines, différentes stratégies de fine-tuning peuvent

être envisagées. Lorsque la similarité entre les domaines source et cible est forte, un simple ajustement de la couche de sortie, en conservant le reste du réseau gelé, peut suffire à obtenir de bonnes performances. À l'inverse, si le domaine cible diffère sensiblement en termes de distribution, de structure des données ou de sémantique , il devient nécessaire de réentraîner une partie plus importante du modèle. Un fine-tuning partiel permet alors d'adapter les couches intermédiaires et finales, tout en maintenant les premières couches gelées pour conserver l'extraction de caractéristiques génériques. Dans les cas les plus extrêmes, où le domain shift est prononcé (par exemple entre des images naturelles et images médicales), un fine-tuning complet ou progressif, peut s'imposer afin de réaligner en profondeur les représentations apprises. Cette flexibilité dans la méthode de fine-tuning choisi, permet non seulement de maximiser les performances prédictives, mais aussi de limiter les coûts énergétiques en évitant d'entraîner systématiquement un modèle depuis zéro.

Plusieurs travaux scientifiques ont illustré empiriquement l'impact du domain shift et les bénéfices du fine-tuning. Par exemple, [Li et al.](#) ont étudié la classification de patchs histopathologiques colorectaux à partir d'un modèle Xception pré-entraîné sur ImageNet. Sans adaptation ni réentraînement, les caractéristiques extraites du réseau atteignent déjà 96,4 % d'exactitude, mais restent limitées par la différence marquée entre images naturelles et biomédicales. En appliquant un fine-tuning sur 100 000 images annotées, les auteurs font passer l'exactitude globale à 98,4 %, avec des améliorations notables pour des classes difficiles comme le stroma (de 87 % à 94 %). De plus, ce fine-tuning améliore significativement la capacité du modèle à prédire, à partir d'images de cancer du poumon non utilisées pour l'adaptation, l'expression de gènes immunitaires. Ces résultats confirment que, face à un domain shift, le fine-tuning d'un modèle pré-entraîné est non seulement souhaitable mais souvent indispensable pour exploiter au mieux les connaissances transférées. De même, [Shirokikh et al.](#) ont montré que selon la nature du domain shift, il peut être plus pertinent de n'ajuster qu'un sous-ensemble spécifique de couches du réseau. Leur étude, menée sur la segmentation d'IRM cérébrales provenant de six centres cliniques différents, montre que les variations d'intensité liées aux protocoles d'acquisition affectent fortement les premières couches convolutionnelles. Contrairement aux approches classiques qui privilégient l'ajustement des couches finales, les auteurs démontrent que le fine-tuning des premières couches conduit à de bien meilleures performances de transfer. En particulier lorsque les données annotées du domaine cible sont rares. En effet, dans presque tous les cas testés, cette stratégie surpassé non seulement le fine-tuning des couches finales, mais aussi celui de l'ensemble du réseau lorsque les ressources sont limitées ([Shirokikh et al.](#)). Cela confirme que les couches initiales, bien qu'habituellement considérées comme génériques, peuvent en réalité capturer des informations très spécifiques au domaine, et que leur réajustement ciblé est une réponse efficace aux décalages de bas niveau comme les variations de contraste ou d'intensité. Ces études confirment l'importance d'adapter la stratégie de fine-tuning selon la nature du domain shift.

Dans ce mémoire, l'effet du domain shift n'est pas évalué expérimentalement. Il est introduit pour situer le problème. En résumé la littérature suggère qu'un décalage entre les données source et cible marqué dégrade l'exactitude, ce qui conduit souvent à recourir à un fine-tuning plus profond (dégeler davantage de couches). Ce choix augmente le coût d'entraînement par image (J/image), car la rétropropagation et les mises à jour portent sur un plus grand nombre de paramètres. Ainsi, le nombre de couches dégelées (hyperparamètre du transfer learning) constitue un levier qui oriente la position des solutions sur le front de Pareto exactitude–efficacité et justifie son étude dans notre optimisation multiobjectif.

2.3.2 Le rôle du modèle dans l'équilibre exactitude-efficacité

Le choix du modèle pré-entraîné constitue un facteur déterminant de l'empreinte énergétique globale, tant au niveau de l'inférence que de l'entraînement. Il dépend à la fois de considérations techniques et de la nature de la tâche cible. Il doit être pensé en lien étroit avec la stratégie de transfert learning retenue, elle-même conditionnée par plusieurs facteurs.

La taille du jeu de données cible est un autre facteur à prendre en compte. Lorsqu'elle est réduite, les possibilités de fine-tuning profond sont limitées, tant pour des raisons d'overfitting que de ressources. Dans ce cas, des approches plus conservatrices, comme la feature extraction, sont généralement préférées, car elles mobilisent moins de capacité de calcul et permettent de limiter l'empreinte énergétique liée à l'entraînement.

Le choix du modèle préentraîné est guidé par plusieurs éléments : (i) les ressources matérielles disponibles telles que la RAM, le GPU ou le budget énergétique, qui contraignent la profondeur du réentraînement et le type de modèle mobilisable ; (ii) la complexité des données cibles, qui détermine la capacité requise du modèle à représenter les structures pertinentes ; (iii) la stratégie de transfert learning choisie, qui dépend directement des deux premiers éléments ; (iv) les contraintes de déploiement, comme la latence, la mémoire ou la consommation énergétique, particulièrement critiques dans des environnements embarqués ou mobiles. Ainsi, des modèles légers comme MobileNet ou EfficientNet sont généralement privilégiés pour les applications sur smartphone, tandis que des modèles plus lourds comme ResNet ou DenseNet sont mieux adaptés aux systèmes centralisés disposant de ressources suffisantes.

Un autre point important à considérer est que la taille d'un modèle ne reflète pas nécessairement sa complexité réelle ni son efficacité énergétique. Certains modèles bénéficient d'architectures plus optimisées, ce qui leur permet d'obtenir de meilleures performances à taille égale, voire supérieure, par rapport à des modèles plus anciens. Les travaux de Desislavov *et al.* ont montré que les modèles dits *state of the art* sont souvent suivis de variantes offrant des performances comparables tout en étant plus économies en ressources. Le cas d'EfficientNet en est une illustration. Ce modèle démontre qu'il est possible d'améliorer simultanément la précision et l'efficacité énergétique en repensant la structure architecturale. Cette tendance à l'optimisation des architectures s'est accentuée au fil du temps, conduisant à des modèles capables d'atteindre des niveaux de performance équivalents, voire supérieurs, avec un nombre de paramètres ou de FLOPs réduit.

Ces éléments sont fortement interdépendants. Le modèle, la stratégie de TL et les objectifs d'optimisation ne peuvent être décidés séparément. Leur bon ajustement conditionne à la fois la performance finale et l'efficacité énergétique globale de l'approche. Ce choix initial structure également l'espace d'optimisation des hyperparamètres, et constitue ainsi une étape déterminante dans la recherche d'un compromis entre exactitude et efficacité.

2.3.3 Évolutions récentes

Introduite par Hinton *et al.*, la distillation de connaissances consiste à transférer l'expertise d'un modèle "professeur" volumineux vers un modèle "étudiant" plus léger, tout en préservant des performances proches de celles du modèle d'origine. Cette technique rend le déploiement possible sur des appareils contraints grâce à l'architecture compacte de l'étudiant. La méthode repose principalement sur l'utilisation de *soft targets*, c'est-à-dire la distribution de probabilité prédite par le professeur pour chaque classe. Contrairement aux *hard targets* (la seule étiquette

correcte), ces vecteurs de probabilités offrent plus d'informations. Cette approche permet une meilleure généralisation en apprenant les relations entre les classes. Par exemple, si le professeur associe une image à 70 % à la classe «chien» et à 20 % à la classe "ours", l'étudiant comprend que "chien" et "ours" partagent des similarités, ce qu'une simple étiquette "chien" n'aurait pas permis. La fonction de perte combine alors deux composantes. La divergence entre les soft targets du professeur et celles de l'étudiant, et l'erreur sur les hard targets. Pour faciliter cet apprentissage, une température $T > 1$ est appliquer pour lisser la distribution des probabilités.

Lors de l'entraînement, l'étudiant ajuste ses poids pour reproduire simultanément la distribution lissée (soft targets) et l'étiquette exacte (hard target). Bien que l'ensemble de données utilisé soit généralement identique à celui du professeur, l'article montre qu'un sous-ensemble peut suffire pour obtenir d'excellentes performances. Par exemple, sur MNIST, un étudiant n'ayant jamais vu le chiffre « 3 » pendant la phase d'apprentissage atteint 98,6 % de précision sur cette classe après distillation, contre 99 % lors d'un entraînement classique sur l'ensemble complet. De même, sur CIFAR-10, un étudiant formé sur seulement 50 % des images conserve plus de 90 % de la précision du professeur. Des expérimentations sur MNIST confirment ce gain. Le modèle professeur ne commet que 67 erreurs, l'étudiant entraîné de zéro en effectue 146, contre seulement 74 lorsqu'il a bénéficié de la distillation. De plus, [Hong et al.](#) ont montré que la distillation, combinée à une compression en largeur, c'est-à-dire une réduction du nombre de canaux par couche, permet de réduire la taille d'un modèle MobileNetV1 de 42,27 % tout en préservant une grande partie de sa précision.

Bien que la distillation puisse s'apparenter à une forme de transfer learning, elle se différencie par son objectif et ses mécanismes d'apprentissage. La distillation vise à compresser un modèle pour faciliter son déploiement sur des appareils moins puissants ou pour diminuer les coûts d'un déploiement à grande échelle. En revanche, le transfer learning vise à adapter le modèle à une nouvelle tâche cible. Néanmoins, ces deux méthodes utilisent les connaissances apprises au préalable pour l'apprentissage du nouveau modèle. Dans le cas de la distillation, la tâche et le domaine restent généralement identiques, contrairement au transfer learning qui implique souvent un changement de tâche ou de domaine. L'apprentissage pour ce dernier vise à exploiter les représentations déjà apprises dans le but de réduire le temps d'entraînement et de tirer parti des performances des modèles préentraînés. Ces méthodes sont plus complémentaires qu'identiques, mais elles permettent toutes deux de tirer parti de connaissances préalablement intégrées. Par exemple, [Yang et al.](#) ont montré que l'utilisation conjointe des mécanismes de transfer learning et de distillation peut améliorer significativement les performances des modèles compacts. Avec leur méthode "Attention and Feature Transfer-based Knowledge Distillation" (AFT-KD), ils transforment non seulement les caractéristiques finales du modèle professeur. Ils intègrent également les représentations intermédiaires et les cartes d'attention. Cette approche permet à l'étudiant de bénéficier à la fois des connaissances globales et locales du modèle professeur. Cela optimise l'apprentissage et réduit la perte de précision lors de la compression.

Le pruning, est une autre technique de compression qui vise à réduire le nombre de paramètres d'un modèle. Le pruning fonctionne en retirant les neurones ou paramètres redondants. Ceux-ci ne participent pas de manière significative à l'amélioration des résultats. Les paramètres peuvent inclure : des poids nuls ou similaire. Comme la distillation cette technique améliore diminue la complexité computationnel et permet de déployer des modèles sur des environnements contraints. Si le pruning détériore la précision en supprimant des informations importante, le modèle peut toujours être récupérer grâce au fine-tuning. En général, l'élagage entraîne une perte minime de la précision ([Liang et al.](#)).

La quantization est une technique semblable au pruning dans son objectif de diminuer l'intensité computationnelle à l'inférence. Elle consiste à réduire la précision des données manipulées, notamment les poids et les activations. Elle remplace les nombres flottants 32 bits en entier 8 bits, ce qui permet de réduire la mémoire nécessaire pour utiliser le modèle et accélère les calculs. Néanmoins, une petite perte de précision peut survenir, mais il existe des approches qui minimisent cet impact comme la quantization sensibilisée à l'entraînement. Les LLM recourent très souvent à la quantization surtout pour faciliter le déploiement, car ils permettent de diminuer la taille des modèles de manière significative. Par exemple avec la méthode de quantization après la phase d'entraînement il a été possible de réduire la taille d'un modèle (GPTQ) de 14 Go à environ 2 Go après une quantization 4-bits. Ce qui équivaut à une diminution de 80 % de la taille du modèle ([Roy](#)).

En résumé, ces techniques récentes, telles que la distillation, le pruning ou la quantization, favorisent le développement de modèles à la fois plus efficaces et plus performants. Elles sont souvent appliquées de manière complémentaire afin de combiner leurs avantages respectifs. Ces approches permettent notamment de déployer des modèles sur des appareils embarqués disposant de ressources computationnelles limitées. Dans ce contexte, l'optimisation des hyperparamètres joue un rôle essentiel pour exploiter pleinement le potentiel de ces techniques et garantir un compromis satisfaisant entre précision, taille du modèle et coût computationnel.

Bien que ces techniques ne soient pas mises en oeuvre dans le cadre de l'expérimentation présentée, elles constituent des leviers pertinents pour améliorer l'efficacité des modèles et seront considérées comme pistes pour de futurs travaux.

3 Hyperparamètres

Cette section établit le cadre des hyperparamètres utilisés dans ce travail. Elle commence par rappeler la différence entre paramètres et hyperparamètres afin de préciser ce qui est fixé avant l'apprentissage. Elle analyse ensuite la manière dont ces choix orientent le compromis entre exactitude et efficacité énergétique pendant l'entraînement. Les hyperparamètres sont organisés selon leur fonction d'optimisation, de régularisation, de durée d'entraînement et de fine tuning, puis quatre leviers centraux sont détaillés, à savoir nombre d'epochs, taux de dropout, taille de batch et weight decay. La section se termine par la présentation de fANOVA, méthode retenue pour estimer l'importance relative de chaque hyperparamètre et éclairer la recherche multiobjectif.

3.1 Rappel : Distinction entre paramètres et hyperparamètres

Il est important de distinguer clairement les hyperparamètres des paramètres, cette distinction étant essentielle pour la suite du mémoire.

Les paramètres sont les valeurs internes du modèle apprises automatiquement lors de l'entraînement. Ils sont mis à jour à chaque itération. Dans le cas d'un CNN, ces paramètres correspondent aux poids des filtres des couches convolutionnelles ainsi qu'aux poids et biais des neurones dans les couches fully-connected. Ils sont optimisés grâce à la backpropagation. Ce sont ces paramètres qui sont initialisés à partir d'un modèle préentraîné lors d'une tâche de transfer learning. Ils représentent la capacité du modèle à résoudre une tâche donnée. Leur quantité dépend des choix architecturaux du réseau (voir fig. 2) et peut varier de quelques millions à plusieurs centaines de millions dans le cas des modèles les plus complexes.

Les hyperparamètres sont définis manuellement avant l'entraînement par l'utilisateur ou par des techniques d'optimisation automatisée. Ils influencent la structure du modèle, sa manière d'apprendre, ainsi que ses performances globales. Par exemple, la taille des batchs détermine combien d'exemples sont traités avant chaque mise à jour des poids grâce à la backpropagation. Le nombre d'epochs indique combien de fois l'ensemble du jeu de données est parcouru durant l'apprentissage. Les hyperparamètres jouent un rôle clé dans le compromis entre exactitude et efficacité, et capacité de généralisation. Les hyperparamètres sont le plus souvent étroitement liés à la tâche et à l'architecture du modèle. Il n'existe pas de valeurs universelles qui fonctionnent pour tous les cas.

3.2 L'impact des hyperparamètres sur le compromis exactitude-efficacité

Dans une approche multiobjectif exactitude-efficacité, le compromis a pour but de minimiser la consommation d'énergie lors de l'entraînement et de maximiser la performance du modèle. La consommation d'énergie est devenue un élément clé, car la tendance actuelle se dirige vers des modèles toujours plus complexes dans le but d'atteindre les meilleures performances possibles. Cela a comme impact d'augmenter les coûts financiers et de créer des effets environnementaux négatifs. La consommation d'énergie se décompose en temps de calcul qui est lié à l'utilisation intensive des CPU, GPU ou TPU sur la durée tandis que l'utilisation de la mémoire vive consomme de l'énergie pour manipuler et stocker les données. Ces deux composantes influencent directement la consommation d'énergie totale. Les hyperparamètres sont des leviers qui permettent de trouver le compromis optimal entre les objectifs. C'est un compromis, car l'amélioration de l'efficacité énergétique et l'amélioration de l'exactitude sont contradictoires. En règle générale les

architectures plus complexes favorisent une meilleure exactitude, mais consomment aussi plus de ressources par image. Face à cette contradiction, l'enjeu est de trouver un équilibre. C'est le principe de l'optimisation multiobjectif, qui postule qu'il est souvent impossible de trouver une solution unique, parfaite pour tous les objectifs à la fois. Le but est donc de générer un ensemble de solutions représentant les meilleurs compromis possibles, que l'on appelle le front de Pareto. Le front de Pareto permet à un praticien de choisir la solution la plus adaptée à ses besoins. Une solution est optimale (Pareto optimale) si aucune solution ne la domine.

Pour illustrer ce compromis, considérons l'influence de la taille des batchs. Augmenter leur taille peut diminuer le temps d'entraînement par epochs grâce à une meilleure parallélisation matérielle, ce qui augmente le débit d'image par Joules consommé. Mais cet ajustement à la hausse augmente la pression sur la mémoire et peut comme l'on montrer l'étude de [Kandel et Castelli](#), nuire à la capacité de généralisation du modèle. La taille du batch dépend aussi fortement de la capacité du matériel à pouvoir garder un nombre d'exemples en mémoire. D'autres hyperparamètres dépendent aussi de contraintes matérielles ou financières et ont un espace d'ajustement limité. Ces exemples démontrent l'existence d'un espace de compromis subtil entre les hyperparamètres pour obtenir un bon ratio entre performance et efficacité.

En plus des hyperparamètres qui impactent directement les ressources d'autres influencent la dynamique d'apprentissage du modèle, par conséquent la précision. Le learning rate est le paramètre le plus critique lors de l'entraînement. Il détermine la vitesse à laquelle le modèle met à jour ses paramètres à chaque itération et influence directement la stabilité de convergence vers une solution optimale. Un taux mal ajusté peut ralentir la convergence, imposant davantage d'epochs et réduisant ainsi la performance obtenue par unité d'énergie consommée. Il peut aussi provoquer la divergence du modèle, rendant l'énergie dépensée inutile. Donc un learning rate mal réglé diminue le rendement par Joule, parce que chaque Joule dépensé produit moins de gain en performance. À cela s'ajoute le choix de l'optimizer comme Adam et SGD. Il interagit fortement avec le learning rate, car il représente la technique par laquel sont mis à jours les paramètres. Enfin, les techniques de régularisation comme le dropout, weight decay visent spécifiquement à améliorer la généralisation en évitant le sur-apprentissage. Cela permet au modèle d'arriver à des performances convenables plus rapidement. Ce qui permet d'éviter des entraînements inutilement long. L'ajustement de ces hyperparamètres est donc une stratégie d'optimisation à part entière, où l'objectif est de maximiser l'efficacité de chaque epoch pour réduire la durée totale de l'entraînement.

La complexité de cette optimisation réside dans les fortes interdépendances entre ces différents hyperparamètres, qui ne peuvent être ajustés isolément. Par exemple, un praticien cherchant à réduire le temps de calcul par image pourrait augmenter la taille des batchs, diminuer le nombre de couche gelées et utiliser un learning rate plus élevé pour accélérer la convergence. Cependant, cette stratégie peut se révéler contre-productive. Comme mentionné, une grande taille de batch peut nuire à la généralisation ([Kandel et Castelli](#)), tandis qu'un learning rate élevé combiné à de grands batchs peut rendre l'entraînement instable et empêcher le modèle d'atteindre un minimum optimal. Pour corriger la perte de performance qui en résulte, il faudrait alors introduire des techniques de régularisation plus fortes ou, paradoxalement, augmenter le nombre d'epochs, annulant ainsi le gain énergétique initialement visé par une amélioration de l'efficacité. Cet exemple illustre que l'espace de recherche des hyperparamètres est un système complexe aux interactions non linéaires. Son exploration manuelle est donc difficile et souvent sous-optimale, ce qui justifie le recours à des stratégies d'optimisation d'hyperparamètres automatisées pour décider efficacement ces compromis et identifier des solutions proches du front de Pareto.

En résumé, rechercher un compromis entre exactitude et efficacité revient à résoudre un problème dans un espace d'hyperparamètres souvent large. Les effets sont fortement interdépendants et non linéaires. Une recherche manuelle ou fondée uniquement sur des approches empiriques risque de conduire à des solutions sous-optimales, à des réglages peu généralisables et à une consommation énergétique excessive. L'optimisation automatisée permet d'explorer et d'exploiter l'espace de manière systématique et reproductible, en produisant un ensemble de compromis non dominés plutôt qu'un choix isolé, laissant la décision finale s'adapter aux contraintes comme J/image, seuils de performance, etc.

3.3 Catégories d'hyperparamètres

Avant d'aborder toute stratégie d'optimisation multiobjectif, il convient de souligner que les hyperparamètres ne constituent pas un ensemble homogène, chacun influence des aspects complémentaires du modèle et requiert des méthodes d'ajustement adaptées. Plusieurs classifications des hyperparamètres existent dans la littérature, notamment en fonction de leur effet sur la structure du réseau, l'optimisation ou la régularisation ([Pramoditha](#)). Dans le cadre de ce mémoire, est proposé une organisation fonctionnelle adaptée à l'optimisation multiobjectif exactitude-efficacité, répartie en quatre catégories : optimisation, régularisation, durée d'entraînement et fine-tuning. Cette section a pour objectif de présenter ces grandes catégories afin de poser les bases nécessaires à la compréhension des hyperparamètres détaillés dans les sections suivantes.

- **Hyperparamètres liés à l'optimisation** : ils influencent directement la dynamique d'apprentissage. Cela inclut le learning rate, le choix de l'*optimiseur* (comme SGD ou Adam), la weight decay, ou encore la taille des batchs. Ces paramètres affectent à la fois la stabilité, la vitesse de convergence et la consommation de ressources.
- **Hyperparamètres liés à la régularisation** : ils visent à améliorer la capacité de généralisation du modèle tout en réduisant le risque de surapprentissage. Le plus courant est le *dropout rate*, mais d'autres techniques existent (dropout, weight decay, etc.).
- **Hyperparamètres liés à la durée d'entraînement** : le principal ici est le nombre d'epochs, qui agit directement sur le temps de calcul et donc sur la consommation énergétique globale. Un mauvais réglage peut provoquer un sous-apprentissage ou un surapprentissage inutilement coûteux.
- **Hyperparamètres spécifiques au transfer learning** : dans un contexte de transfer learning, le nombre de couches gelées ou la décision de réentraîner certaines couches intermédiaires représentent également des leviers importants pour optimiser le compromis précision-énergie.

Il existe une autre catégorie importante "Hyperparamètres architecturaux" qui définit la structure du modèle. Elle comprend le nombre de couche convolutionnel, le nombre de filtre par couche, les fonctions d'activations, etc. Elles ne sont pas revues dans le cadre du mémoire car la structure d'un modèle n'est généralement pas modifiable pour le transfer learning.

3.4 Hyperparamètres clés de l'entraînement

Parmi l'ensemble des paramètres identifiés certains jouent un rôle clé dans le compromis performance-énergie. Les hyperparamètres sont définis pour comprendre comment leur fonctionnement impacte individuellement l'entraînement lors de l'expérimentation.

Epoch L'epoch correspond au nombre de fois où un jeu de données complets va traverser les couches du modèles en feedforward et pour la back propagation. L'objectif est de trouver un nombre d'epoch qui n'arrête pas l'entraînement trop tôt ou trop tardivement. Le risque est un sous-apprentissage ou un sur-apprentissage et des traitements computationnel inutiles.

Il n'existe pas de méthode factuelle permettant de déterminer le nombre d'epochs optimal avant l'entraînement. Le nombre diffère d'un jeu de donnée à un autre comme l'ont présenté dans leur recherche [Afaq et Rao](#). L'attention doit être portée sur la perte d'entraînement et la perte de validation, il démontre que tant que ces deux indicateurs diminuent ensemble, l'entraînement est efficace. Il devient nécessaire de stopper l'entraînement dès que l'erreur de validation augmente, car elle annonce un surapprentissage.

Stopper l'entraînement avant que le nombre d'epochs initialisé soit atteint est une méthode appelé l'*early stopping*. Cependant, comme le présente M. [Prechelt](#), les courbes de validation sont rarement lisses. Elles fluctuent et présentent les multiples minima locaux. Une règle simpliste d'arrêt qui viserait à stopper le modèle à la moindre hausse de la perte de validation, pourrait interrompre l'entraînement prématurément. Cela risque d'avoir comme implication de manquer un minimum global ultérieur qui offrirait une meilleure généralisation. Ainsi, l'*early stopping* peut être considéré comme un problème d'optimisation visant à trouver le meilleur compromis entre la performance du modèle et les ressources (temps, énergie) engagées.

Batch size La taille d'un batch correspond au nombre d'échantillons qui traversent le réseau avant la mise à jour des paramètres, à travers la phase de feedforward et de backpropagation. Le choix de la batch size n'est pas tout à fait libre, il dépend de plusieurs contraintes, notamment les capacités matérielles pour stocker un grand nombre d'images en mémoire, la résolution des images qui augmente la charge mémoire, ainsi que le nombre total d'échantillons disponibles.

Utiliser une taille de batch trop élevé peut rallonger le temps de convergence sans forcément améliorer la précision. Un batch trop petit peut aussi nuire aux performances. Le nombre d'échantillons disponibles reste un facteur limitant, en particulier dans les contextes de transfert learning, comme en recherche médicale, où l'on dispose parfois de seulement quelques centaines d'exemples.

L'étude de [Magboo et Magboo](#) illustre bien les défis liés à ce paramètre. La théorie est bien posée, mais son application se heurte souvent à des contraintes pratiques. Malgré cela, le réglage du batch size reste crucial. Les auteurs montrent que, pour une même architecture et un matériel identique, certaines valeurs de batch améliorent bel et bien la précision du modèle.

La recherche souligne aussi l'importance de l'interaction entre le batch size et le learning rate. Une étude de [Usmani et al.](#) a montré que ces deux hyperparamètres ont un impact important non seulement pris séparément, mais aussi combinés. Grâce à une analyse ANOVA, les auteurs ont mis en évidence la nécessité de trouver un bon compromis entre les deux. Par exemple, pour une tâche de classification de tumeurs cérébrales, la meilleure performance de 99,56% a été obtenue avec un batch size de 32 associé à un learning rate de 0,01. Il ne faut donc pas considérer le batch size comme un paramètre à régler isolément, mais comme le résultat d'un équilibre avec le taux d'apprentissage.

Learning rate Le learning rate détermine l'amplitude des ajustements appliqués aux poids du modèle à chaque itération. En s'appuyant sur les résultats du gradient de la fonction de perte. Un taux d'apprentissage trop élevé peut entraîner une convergence instable, où le modèle "saute"

par-dessus les minima optimaux, risquant de ne jamais converger. Dans le pire des cas, il peut provoquer une divergence, rendant l'énergie et le temps de calcul dépensés totalement inutiles. À l'inverse, un taux trop faible ralentit considérablement l'apprentissage, car les ajustements des poids sont minimes..

Le learning rate conditionne les performances (exactitude) globales de l'entraînement. En pratique, un LR bien réglé accélère la convergence vers une solution performante et réduit l'énergie et le temps nécessaires pour atteindre une exactitude cible. Il est aussi important de noter que l'efficacité du learning rate est directement liée au choix de l'optimiseur, tel que Adam, AdamW et SDG, et à la taille des batchs. Certains optimiseurs adaptatifs permettent de compenser en partie un mauvais réglage du learning rate. Cependant, cette compensation ne permet pas de ne pas choisir le taux avec précision.

Comme l'ont montré [Usmani et al.](#), une analyse de l'interaction entre le learning rate et le batch size est cruciale pour atteindre des performances optimales. Trouver le bon équilibre entre ces hyperparamètres est donc essentiel pour naviguer le compromis exactitude-efficacité.

Weight decay La weight decay, précédemment associé à la régularisation L_2 , est une technique visant à limiter le surapprentissage. Elle réduit progressivement la valeur des paramètres pour les rapprocher de zéro. Cet effet a pour objectif d'empêcher le modèle de s'appuyer excessivement sur une caractéristique spécifique pour produire ses prédictions. Elle agit en modifiant directement la mise à jour des paramètres dans l'optimiseur. Cela a pour conséquence de réduire la complexité du modèle et d'améliorer la généralisation. En pratique, les paramètres ayant de grandes valeurs vont diminuer plus rapidement, car la réduction est proportionnelle à leur valeur. Les paramètres proches de zéro vont très peu changer, car la pénalisation est faible pour eux. Si un paramètre est réellement important pour la tâche, alors lors de la gradient descent, il va à nouveau croître, compensant cet effet de décroissance, tandis que les paramètres les moins utiles vont dériver vers zéro avec le temps. Alors, ceux réellement informatifs vont se stabiliser à une valeur. L'impact de la weight decay se manifeste à travers le choix d'un coefficient de décroissance. Ce coefficient influence la dynamique d'apprentissage : trop faible, l'effet est négligeable et trop élevé, le modèle risque un sous-apprentissage, car les paramètres sont trop petits pour bien représenter la complexité de la tâche.

Contrairement à la régularisation L_2 classique, qui ajoute un terme de pénalisation dans la fonction de perte, la weight decay proposée par [Loshchilov et Hutter](#) applique une réduction des poids séparément du calcul du gradient. Cette séparation évite l'interaction indésirable entre la pénalisation et les mécanismes d'adaptation du learning rate présents dans l'optimiseur AdamW, ce qui améliore la stabilité et la régularité de la décroissance des poids.

Dropout rate Le *dropout rate* est une technique de régularisation qui consiste à désactiver aléatoirement un pourcentage de neurones pendant l'entraînement, réduisant ainsi le nombre de paramètres actifs à chaque itération. Par exemple, dans EfficientNetV2, le dropout rate est fixé à 20 %. Cette méthode vise à limiter la dépendance du réseau à un trop petit nombre de neurones ou à des connexions spécifiques. Cela contribue ainsi à réduire les effets du surapprentissage. À chaque itération durant l'entraînement, les neurones désactivés ne participent pas aux calculs des activations ni à ceux des gradients. Cependant, en phase d'inférence, aucun neurone n'est désactivé, ce qui permet d'utiliser pleinement la capacité du réseau.

Le *dropout rate* s'applique différemment aux CNN et aux RNN. Dans les CNN, il est souvent placé uniquement dans la tête de sortie du modèle, c'est-à-dire dans les couches fully

connected. Dans les RNN, en revanche, il peut être appliqué entre les couches récurrentes elles-mêmes, afin de limiter le surapprentissage tout en préservant la capacité du réseau à modéliser les dépendances séquentielles.

Les hyperparamètres présentés n'influencent pas directement la consommation d'énergie du modèle, mais ils permettent, pour un coût énergétique équivalent, d'améliorer la dynamique d'apprentissage, ce qui se traduit par un meilleur rendement en termes d'exactitude par unité d'énergie (efficacité), l'objectif central de ce travail. Toutefois, la batch size constitue un hyperparamètre ayant un impact conjoint sur la convergence du modèle et sur sa consommation énergétique. Leur étude s'inscrit donc pleinement dans la démarche visant à optimiser simultanément la précision et l'efficacité énergétique, et justifie leur intégration et leur évaluation expérimentale dans ce mémoire.

3.5 Méthode fANOVA

Dans le but d'identifier les hyperparamètres ayant le plus d'influence sur chaque objectif, la méthode *fANOVA* (*functional ANOVA*) proposée par [Hutter et al.](#) constitue une approche pertinente. Elle permet d'estimer l'importance individuelle d'un hyperparamètre ou d'un groupe d'hyperparamètres en analysant leur contribution à la variance des performances du modèle. Concrètement, la méthode consiste d'abord à entraîner un modèle plusieurs fois, avec différentes configurations d'hyperparamètres. Le but est d'obtenir un ensemble de résultats empiriques. Un modèle de substitution, généralement une *random forest*, est ensuite ajusté pour approximer la fonction de performance sur l'ensemble de l'espace d'hyperparamètres. Ce modèle sert à interpoler les performances attendues pour des configurations non testées, en s'appuyant sur celles observées. À partir de cette approximation, fANOVA calcule les "marges conditionnelles". Il s'agit de la performance moyenne attendue d'un hyperparamètre H_1 , lorsque les autres hyperparamètres (H_2, H_3, \dots) varient librement sur leur domaine. Cette fonction marginale permet de mesurer l'effet isolé de H_1 . Finalement, l'importance de l'hyperparamètre est estimée comme la variance de cette fonction marginale, c'est-à-dire la quantité de variance expliquées par H_1 dans la performance du modèle.

fANOVA n'est pas limitée à la métrique d'exactitude. Elle peut s'appliquer à toute fonction de performance continue. Il peut s'agir d'une métrique de classification (F1, recall, précision), d'une métrique liée à la consommation de ressources (J/image, mémoire, énergie, etc.), ou encore d'un score multiobjectif obtenu par pondération ([Theodorakopoulos et al.](#)). Cependant, fANOVA reste fondamentalement conçue pour des problèmes mono-objectifs, car elle ne permet pas d'estimer directement l'importance des hyperparamètres sur plusieurs objectifs simultanément. Pour répondre à cette limitation, une méthode dérivée a été proposée par [Theodorakopoulos et al.](#), visant à mesurer l'importance des hyperparamètres dans un contexte multiobjectif. Cette approche repose sur une scalarisation des objectifs, à travers une pondération dérivée du front de Pareto. Les auteurs ont expérimenté cette technique sur différentes paires d'objectifs, notamment la paire exactitude/consommation d'énergie. Cette méthode permet d'identifier les hyperparamètres pertinents pour certains compromis entre objectifs, en visualisant l'évolution de leur importance en fonction des poids attribués à chaque objectif. Toutefois, cette approche n'est pas encore accessible via les outils standard d'optimisation tels qu'Optuna. En pratique, utiliser fANOVA séparément sur chaque objectif reste utile. Cela permet d'identifier les hyperparamètres les plus influents pour chacun d'eux, et ainsi d'aider à arbitrer selon les priorités du problème.

Une application concrète de cette logique est proposée par [Wojciuk et al.](#). Ils utilisent

fANOVA pour analyser l'importance des hyperparamètres dans un contexte de classification d'images avec transfer learning. Bien que leur objectif principal soit l'optimisation des performances prédictive, les auteurs soulignent que cette méthode pourrait tout aussi bien s'appliquer à d'autres métriques, telles que la consommation d'énergie ou l'impact sur la mémoire. Ils montrent notamment que le learning rate, la *taille d'image* et le *dropout* sont les hyperparamètres les plus déterminants pour l'exactitude sur leurs différents jeux de données, tandis que des paramètres comme l'optimiseur ou la taille de batch ont une influence négligeable. Une telle analyse séparée, répétée pour chaque métrique pertinente, permet ainsi de guider les décisions d'optimisation en fonction des priorités spécifiques à chaque tâche. fANOVA quantifie donc la contribution de chaque hyperparamètre aux objectifs, répondant ainsi à la sous-question sur les facteurs qui influencent le plus chaque objectif.

4 Métriques : performance et efficacité énergétique

Cette section définit les métriques retenues pour évaluer les modèles et comparer les configurations. Elle couvre la performance prédictive, l'empreinte computationnelle et l'efficacité énergétique afin d'établir un cadre d'évaluation cohérent. Cela servira de base à la construction du front de Pareto et à l'analyse des résultats.

4.1 Indicateurs de performance pour les CNN

Avant de passer aux méthodes multiobjectifs, rappelons les principaux indicateurs de performance utilisés pour évaluer un CNN.

TABLE 2: Panorama des métriques les plus courantes

Métrique	Abrév.	Usage dominant
Exactitude top-1	Acc	Vue d'ensemble rapide, jeux de données équilibrés
F1 macro	F1 _{macro}	Classes déséquilibrées
Top-k	Acc _{top-k}	Évaluer si la vraie classe figure parmi les k prédictions les plus probables
Précision	P	Évalue la proportion de vrais positifs parmi les prédits positifs
Recall	R	Évalue la proportion de vrais positifs détectés
Mean average precision	mAP	Détection d'objets

Exactitude L'exactitude (*accuracy*) est utilisée lorsque les classes sont globalement équilibrées et que l'on cherche d'abord à maximiser la proportion de prédictions correctes, il reste l'indicateur de référence. Il fournit un premier aperçu rapide des performances prédictives globales d'un CNN. Cet indicateur peut cependant être moins approprié, si les classes du jeu de données sont déséquilibrées. Dans un tel cas, le modèle risque d'avoir une exactitude générale convenable, mais cacher un taux d'erreur élevé sur certaines classes moins représentées. Ce biais risque d'entraîner une dégradation des performances réelles à l'inférence si une classe est amenée à être plus présente que lors de l'entraînement. L'exactitude doit donc être complétée avec des métriques moins sensibles au déséquilibre pour refléter plus fidèlement la performance sur l'ensemble des classes.

L'exactitude est définie comme :

$$\text{Acc} = \frac{TP + TN}{TP + TN + FP + FN} \quad (4.1)$$

où

- TP : nombre de vrais positifs ;
- TN : nombre de vrais négatifs ;
- FP : nombre de faux positifs ;
- FN : nombre de faux négatifs.

F1-macro Lorsque l'on travaille sur des jeux de données déséquilibrés, il est important de combiner précision et rappel afin de ne pas privilégier l'une au détriment de l'autre. Le F1-score, défini comme la moyenne de la précision et du rappel. Elle offre une mesure qui pénalise les déséquilibres extrêmes entre ces deux composantes.

Dans une configuration avec plus de deux classes, le F1-macro consiste à calculer le F1-score pour chaque classe considérée séparément, puis à en faire la moyenne arithmétique non pondérée. Cette approche accorde un poids identique à chaque classe, indépendamment de sa taille, ce qui permet de mieux refléter les performances sur les classes minoritaires. Dans le cadre du transfer learning ce problème est souvent rencontré, les disparités entre le nombre d'exemples par classes peut-être important.

Pour une classe donnée, le F1-score est défini par :

$$F1 = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \quad (4.2)$$

où

- Precision = $\frac{TP}{TP+FP}$: proportion de prédictions positives correctes ;
- Recall = $\frac{TP}{TP+FN}$: proportion de cas positifs correctement détectés.

Le F1-macro est alors donné par :

$$F1_{\text{macro}} = \frac{1}{N} \sum_{i=1}^N F1_i \quad (4.3)$$

où N est le nombre total de classes.

Top-k Lorsque l'on travaille avec un jeu de donnée avec beaucoup de classe (ex. ImageNet), et visuellement proche, l'indicateur top-k peut être une solution pour mesurer la capacité de classement du modèle même si il n'est pas parfait. Concrètement, si la vrai classe fait partie des k prédictions les plus probable, cela est considéré comme une prédiction réussie. Cette méthode rend le modèle plus tolérant à l'erreur. La top-k est définie comme :

$$\text{Acc}_{\text{top-}k} = \frac{1}{N} \sum_{i=1}^N \mathbf{1}\{y_i \in f(x_i)[0:k]\} \quad (4.4)$$

où

- N : nombre d'exemples ;
- y_i : l'étiquette de l'exemple x_i ;
- f : la fonction de prédiction (le CNN) ;
- x_i : l'exemple i appartenant à X , le jeu de données ;
- $[0:k]$: indices des k classes les plus probables renvoyées par $f(x_i)$.

Les métriques permettent d'obtenir des informations importantes sur les performances d'un modèle. Cependant, leur interprétation dépend fortement du protocole d'évaluation utilisé. Un modèle peut sembler performant sur un jeu de validation, tout en étant en réalité surspécialisé à ces données. Dans ce cas, ses performances sur des exemples réels ou légèrement différents peuvent chuter. Il est donc essentiel de choisir une métrique adaptée au problème à résoudre, mais aussi de veiller à la rigueur du protocole d'évaluation pour s'assurer que le modèle généralise

bien. Par exemple dans le cadre d'une détection de pathologie cancérigène, des chercheurs ont testé si un modèle de prédiction pouvait maintenir de bonne performance dans de nouvelles conditions. Ils ont utilisés un modèle pour détecter les métastases des ganglions du cancer du sein qui obtenait de très bon résultat sur son jeu de validation en interne. Lorsque le modèle a été appliqué sans réadaptation à des ganglions d'indication chirurgicale différente sa sensibilité à chuter drastiquement la FROC³ passe à 0.503 (contre > 0.9 en validation interne). La seule façon de récupérer la performance a été de réentraîner le modèle avec un échantillon local de ces nouveaux ganglions ([Jarkman et al.](#)).

Hormis les indicateurs précédemment présentés, plusieurs autres métriques sont couramment utilisées pour évaluer la performance des CNN, même si elles ne seront pas exploitées dans l'expérimentation : la précision (*precision*), qui s'avère essentielle lorsque la classe positive est rare (par exemple pour le filtrage de spam ou la détection de fraude) car elle mesure la proportion de vrais positifs parmi l'ensemble des prédictions positives et évite d'obtenir une précision artificiellement élevée tout en ne repérant aucun cas réel. Le rappel (*recall*), qui évalue la capacité du modèle à identifier correctement l'ensemble des exemples positifs et à limiter les absences de faux négatifs. Enfin la mAP (mean Average Precision), norme de référence en détection d'objets, qui agrège précision et rappel sur plusieurs seuils de confiance pour fournir un score unique facilitant la comparaison entre modèles. Cette courte présentation illustre la diversité des métriques d'évaluation disponibles pour caractériser la qualité d'un CNN.

En optimisation multiobjectif, la métrique retenue oriente directement la recherche d'hyperparamètres. Chaque critère : F1-score pour un jeu déséquilibré, exactitude pour une vue d'ensemble, top-k pour un grand nombre de classes, etc., répondent à un enjeu spécifique. Le choix de la métrique est donc déterminant, un algorithme qui optimise un indicateur inadapté risque de produire un modèle inutilisable en conditions réelles. Il est donc préférable en fin d'entraînement d'utiliser plusieurs métriques afin d'avoir une vision complète des performances prédictives du modèle.

4.2 Indicateurs d'empreinte computationnelle

Ces dernières années, la taille et la complexité des réseaux de neurones convolutifs ont fortement augmenté, conduisant à un besoin en calcul toujours plus important. Pour évaluer non seulement la qualité des modèles (précision, F1, etc.), mais aussi leur impact environnemental et économique, il est devenu essentiel de quantifier leur empreinte computationnelle. Les métriques FLOPs (nombre total d'opérations flottantes) et FLOPS (débit d'opérations par seconde) se sont ainsi imposées comme standards pour comparer, optimiser et analyser leur utilisation en ressource pour les CNN.

Un FLOP décrit une opération arithmétique élémentaire ($+, -, \times, \div$) sur un nombre à virgule flottante. Un nombre à virgule flottante est une façon standardisée de représenter un nombre dans un ordinateur. La virgule peut se déplacer (flotter) selon la puissance de 10. Un nombre flottant peut introduire des erreurs de précision dans les calculs, car ils ne sont pas toujours exacts. Cela vient du fait que certains nombres ne peuvent pas être représentés exactement en binaire.

Les FLOPs (*floating point operations*) correspondent au nombre total d'opérations flottantes effectuées par un modèle, que ce soit durant l'entraînement ou lors d'une passe d'infé-

3. La "Free Response Operating Characteristic" est une métrique utilisée pour évaluer la performance d'un modèle dans la détection et localisation, particulièrement utilisée en imagerie médicale.

rence. À l'inverse, les FLOPS (*floating point operations per second*) mesurent le débit de calcul, c'est-à-dire le nombre d'opérations flottantes réalisées par seconde. Dans la pratique, on cherche à réduire les FLOPs pour diminuer la quantité de calcul et donc la latence, tout en augmentant les FLOPS afin d'exploiter au mieux l'accélération matérielle. Si l'effort se concentre uniquement sur la réduction du nombre total de calculs sans optimiser le débit, les gains énergétiques peuvent être annulés. De fait, ces deux métriques ne sont pas corrélées : diminuer les FLOPs n'implique pas nécessairement une baisse de la latence, ni une amélioration des performances. [Chen et al.](#) insistent sur la nécessité de réduire non seulement les FLOPs, mais surtout les FLOPS pour obtenir de vrais gains de vitesse. Par ailleurs, [Asperti et al.](#) montrent qu'une couche de convolution ayant une grande dimension spatiale (hauteur \times largeur) mais peu de canaux s'exécute beaucoup plus rapidement qu'une couche de dimensions spatiales réduites avec un grand nombre de canaux, malgré un nombre identique de FLOPs. La raison réside que le comptage brut de FLOPs ne prend pas en compte la capacité du matériel à paralléliser les opérations, ce qui est un facteur clé de l'efficacité computationnelle.

Le nombre d'opération nécessaire pour une couche de convolution classique peut être estimer grâce à la formule :

$$D_K^2 \times M \times N \times D_F^2 \quad (4.5)$$

où D_K^2 est la taille du kernel (carré) ; M le nombre de channel en entré (canaux) ; N le nombre de channel en sortie ; D_F^2 la taille de la feature map en sortie ([Howard et al.](#)). La formule ne prend pas en compte le padding et de la stride qui tout deux influence le nombre de FLOPs nécessaire pour une convolution. Il est important de noter que différentes types de convolution existent comme la *depthwise* et *depthwise + pointwise* qui modifie la façon dont on peut estimer les FLOPs totaux par convolution.

En résumé les FLOPS et les FLOPs sont utilisés dans plusieurs domaines : pour exprimer les capacités de calcul d'un processeur ou d'une carte graphique, évaluer et améliorer l'optimisation arithmétique d'un modèle ou d'en estimer son coût computationnel, car plus le nombre total d'opérations est élevé plus le modèle est exigeant en calcul. Il permet aussi de comparer l'efficacité de différentes architectures à complexité équivalente ([Tan et Le \(2019, 2021b\)](#)) ou encore aider à optimiser l'entraînement ou l'inférence, car réduire les FLOPs peut souvent réduire le temps de calcul et les coûts.

Bien qu'ils ne soient pas utilisés dans notre expérimentation, ils constituent un standard largement reconnu. Cette base théorique permet de comprendre comment ils peuvent servir d'indicateur d'efficacité énergétique.

4.3 Indicateurs d'efficacité énergétiques

Dans le cadre de l'expérimentation, la métrique joules par image (ou joules par échantillon) sera utilisé pour avoir une estimer l'évolution de l'efficacité énergétique lors de l'entraînement. La métrique J/image se définit comme :

$$J/\text{image} = \frac{E_{\text{total}}}{N_{\text{images}}} \quad (4.6)$$

- E_{total} : correspond à l'énergie totale en joules consommée pendant l'exécution d'une tâche ;
- N_{images} : le nombre d'images traité au cours de la tâches.

Elle représente donc l'énergie moyenne nécessaire pour traiter une seule image. L'objectif

est de minimiser sa valeur, car plus la valeur est faible, plus le modèle est énergiquement efficace. De plus, elle permet de comparer l'efficacité énergétique de différents modèles, framework ou configuration d'hyperparamètres indépendamment du nombre total d'itération ou de la durée de l'exécution (Yang et Armour; Li et al.). En intégrant à la fois la puissance consommée en watt et la durée d'exécution elle reflète l'efficacité énergétique réel, contrairement à une mesure de puissance seule (W) ou de temps qui serait incomplète. Si l'on ajoute la mesure de l'énergie totale du système la métrique J/image intègre tous les frais de transformation tel que les pré-traitements, copies du CPU vers le GPU, la consommation RAM, les goulots d'étranglement CPU. Ce que ne prend pas en compte une métrique J/FLOP seul, qui cible l'efficacité algorithmique indépendamment du hardware.

Dans le but d'assurer une comparabilité rigoureuse et reproductible des mesures J/image, plusieurs initiatives de standardisation se sont structurées notamment le benchmark MLPerf Power (Tschand et al.) qui définit un protocole de mesure des échantillons par joule et l'outil Carbon Tracker (Anthony et al.) qui intègre automatiquement les relevés de consommation matérielle et l'empreinte carbone associée. Pour publier une comparaison J/image crédible, il est indispensable de fournir les informations sur : l'architecture matérielle avec la version des CPU/GPU, décrire le protocole de mesure comprenant les outils utilisés, la fréquence d'échantillonnage, indiquer le batch size, le nombre d'epochs, et les versions des frameworks et bibliothèques, ainsi que de documenter les conditions environnementales. Cette transparence méthodologique aide à garantir que les résultats sont reproductibles et comparables entre les études et plateformes.

La métrique J/FLOP mesure l'efficacité algorithmique d'un modèle en calculant l'énergie (en joules) nécessaire pour réaliser une opération en virgule flottante. Elle s'oppose directement à la métrique FLOPs/W, qui indique le nombre d'opérations flottantes par joule consommé : un J/FLOP plus faible ou un FLOPs/W plus élevé signale une meilleure performance algorithmique. Cette mesure est particulièrement utile pour comparer des architectures matérielles à nombre d'opérations identique, car elle isole l'efficacité du code indépendamment du matériel. Cependant, J/FLOP ne reflète pas l'efficacité énergétique globale du système. J/FLOP se définit comme :

$$J/FLOP = \frac{E_{\text{total}}}{N_{\text{FLOP}}} \quad (4.7)$$

où

- E_{total} : correspond à l'énergie totale en joules consommée pendant l'exécution d'une tâche ;
- N_{FLOP} : le nombre d'opération à virgule flottante traité au cours de la tâches.

En privilégiant exclusivement la réduction de l'énergie par opération, on risque de sélectionner des hyperparamètres qui améliorent le J/FLOP mais augmentent l'énergie totale par image. Par exemple l'article de M. Chen montre que en effet le pruning peut éliminer plus de 80% des poids réduisant ainsi fortement le nombre de FLOPs, et par conséquent la J/FLOP. Ce qui a pour effet de rendre l'inférence sur GPU sans optimisation spécifique plus lente que pour un réseau dense augmentant la consommation par image (J/image). Ainsi mon choix s'est naturellement porté vers J/image comme métrique pour guider l'optimisation de l'efficacité énergétique.

En résumé, la métrique J/FLOP mesure l'énergie nécessaire pour exécuter une opération en virgule flottante et varie peu lorsque l'on conserve le même matériel. Dans un contexte de TL geler ou dégeler des couches modifie le nombre total de FLOPs, mais le GPU exécute chaque opération avec une efficacité énergétique quasi constante, de sorte que J/FLOP change peu. À l'inverse, J/image profite directement de ce type d'ajustement : en réduisant le nombre de couches dégelées, le temps de traitement par image diminue, ce qui entraîne une baisse proportionnelle de la consommation par image. Ainsi, J/FLOP met en avant l'efficacité computationnelle propre

au matériel et à l'architecture du modèle, tandis que J/image reflète mieux l'impact énergétique concret des choix d'hyperparamètres en transfer learning. C'est pour cette raison que dans ce travail, J/image a été retenu pour optimiser les solutions sur le front de Pareto.

TABLE 3: Liste élémentaire des métriques d'efficacité énergétique

Métrique	Unités	Indique
Joules par image	J/image	Cout énergétique par image
Joules par FLOP	J/FLOP	Efficacité algorithmique du modèle (à minimiser)
FLOPs par Watt	FLOPs/W	Efficacité algorithmique du modèle (à maximiser)
Energy delay product (EDP)	$J \times s$	Compromis temps énergie
samples/sec/Watt	images/s/Watt	Efficacité énergétique opérationnelle
ETA	J (cumulé)	Cout énergétique pour atteindre une précision cible
TTA	s	Temps pour atteindre une précision cible

Hormis, les deux métriques présentées, il en existe d'autres, bien que pas utilisés dans ce mémoire, qui mérite d'être énoncé. Tel que EDP (*energy delay product*) désigne le produit de la consommation d'énergie normalisée et du temps d'execution normalisé. Il reflète le compromis entre la dissipation d'énergie et le retard de performance dans les charges de travail GPU. C'est un indicateur utilisé pour évaluer différentes architectures de modèles en équilibrant la consommation d'énergie et la rapidité de l'exécution ([Vijaykumar et al.](#)). L'EDP accorde une importance équivalente à la dégradation de l'énergie et des performances. Si l'une des deux métriques augmentent, l'EDP augmente également. L'objectif est d'obtenir des valeurs EDP les plus faibles ([Ratković et al.](#)).

Le TTA (*Time to Accuracy*) formulé et popularisé par les auteurs de DAWN Bench ([Coleman et al.](#)), cette métrique vise à permettre une évaluation objective et reproductible entre différents frameworks, matériels, algorithmes et configuration d'hyperparamètres. Elle évalue les performances d'entraînement, en mesurant le cout nécessaire en seconde pour atteindre un seuil spécifié d'exactitude. Ainsi elle intègre les facteurs ayant un impact réel sur les performances globales, en prenant en compte à la fois l'efficacité générale du modèle et matérielle. Cependant cette métrique contient plusieurs inconvénients, elle nécessite de définir a priori un niveau d'exactitude à atteindre, tandis qu'en optimisation multiobjectif est cherché avant tout à explorer le compromis entre exactitude et temps (ou coût énergétique). TTA ne compare que les points qui passent le seuil d'exactitude, il ignore les configurations qui sacrifient un peu d'exactitude pour un gain de temps plus important.

ETA (*energy to accuracy*) est une métrique dérivée de TTA. Elle se différencie en prenant en compte la consommation d'énergie au lieu du temps. Elle peut être obtenue grâce à $TTA \times$ la consommation d'énergie moyenne sur toute la durée de l'entraînement ([You et al.](#)). En résumé ETA est à TTA ce que J/image est à J/FLOP, une métrique plus globale et pratique quand l'énergie est la ressource critique. Cela dit, elle partage les mêmes avantages et inconvénient qu'une métrique basé sur un seuil.

5 Optimisation multiobjectif

Cette section explique les principes utilisés pour optimiser plusieurs objectifs simultanément. Elle commence par présenter la notion de front de Pareto, qui regroupe les solutions offrant les meilleurs compromis possibles. Il sera ensuite montré comment il est possible de choisir un point précis sur ce front selon les besoins. La section décrira aussi le fonctionnement des algorithmes évolutionnaires adaptés aux problèmes multiobjectifs, avec un accent particulier sur NSGA-II. Enfin, il sera présenté comment le framework Optuna est utilisé pour mettre en oeuvre cette optimisation et analyser les résultats grâce à ses outils intégrés.

5.1 Front de Pareto

Introduite par Vilfredo Pareto (1896) dans un contexte économique, la notion d'optimalité de Pareto a depuis été largement adoptée en optimisation à objectifs multiples. Elle est typiquement utilisée pour traiter des problèmes avec plusieurs objectifs contradictoires. Le front de Pareto regroupe les solutions dites optimales qui ne peuvent pas améliorer un objectif sans nuire à un autre. Les optimiseurs d'hyperparamètres multiobjectifs tentent d'approximer le front de Pareto. Dans le cadre de ce mémoire, cela implique de trouver le meilleur compromis entre exactitude et efficacité énergétique dans un espace d'hyperparamètre défini. Les solutions appartenant au front de Pareto ne sont objectivement pas meilleures que les autres, elles incarnent un compromis. Ainsi, le front offre une aide à la décision, un praticien pourra choisir une solution très performante, mais énergivore (un point à une extrémité du front), une solution très efficace, mais avec moins d'exactitude (à l'autre extrémité), ou n'importe quel compromis optimal entre les deux, en fonction de ses contraintes spécifiques. Néanmoins, ces solutions restent supérieures aux autres au sens de la dominance de Pareto ([Yao et al.](#)).

La dominance de Pareto représente la relation entre deux solutions. Une solution X domine une solution Y si elle est meilleure ou égale dans tous les objectifs, et strictement meilleure dans au moins un. Cette définition dépend du sens de l'optimisation choisi (minimisation ou maximisation). Dans le cadre des optimisations multiobjectifs, la notion de contrainte peut être intégrée à la dominance de Pareto. Trois règles sont introduites : (i) une solution X domine une solution Y si elle est réalisable alors que Y ne respecte pas les contraintes. (ii) Si X et Y sont toutes deux irréalisables, X domine Y si la violation globale des contraintes est moindre. (iii) Si les deux solutions X et Y sont toutes les deux réalisables, alors on utilise la règle de dominance de Pareto classique ([Deb et al.](#)). La réalisabilité vise à conserver uniquement les solutions qui respectent les contraintes imposées par le problème. La priorité est donnée aux solutions réalisables, car même si elles ne demeurent objectivement pas meilleure qu'une solution irréalisable. L'objectif reste de conserver uniquement les solutions applicables dans la pratique, comme la présentée l'algorithme NSGA-II. Par exemple, une contrainte pourrait être de garder certains hyperparamètres dans une plage valide telle que le learning rate lr respecte la condition $0 < lr < 1$.

En ce qui concerne les problèmes sans contrainte, le front optimal de Pareto représente un ensemble de solutions dites Pareto-Optimal. Il constitue un espace dans lequel aucune solution ne peut être améliorée sur un objectif sans en détériorer au moins un autre. À la différence d'un problème avec contrainte où une partie de ce front de Pareto est tronqué pour ne reprendre qu'un sous-ensemble du front original, contenant uniquement les solutions réalisables. Également, il peut être une combinaison d'une partie du front original et de la frontière. La frontière représente la limite entre une solution irréalisable et réalisable. Ces solutions sont souvent optimales, car si la solution s'éloigne de la frontière, elle pourrait s'éloigner de ses objectifs. Le but reste toujours

d'obtenir des solutions réalisables, mais les solutions irréalisables servent de guide pour trouver une solution idéale réalisable. NSGA-II utilise cette approche, si l'algorithme doit comparer deux solutions non réalisables, il préférera celle qui viole le moins les contraintes. Cela crée une "pente" qui guide la recherche hors de la zone interdite et en direction de la frontière ([Deb et al.](#)).

La question n'est donc plus de trouver une solution unique, mais de parvenir à approximer cet ensemble de solutions optimales. Les méthodes d'optimisation multiobjectif, telles que les approches évolutionnaires qui vont être abordées, ont précisément pour but de construire et d'approximer ce front de Pareto.

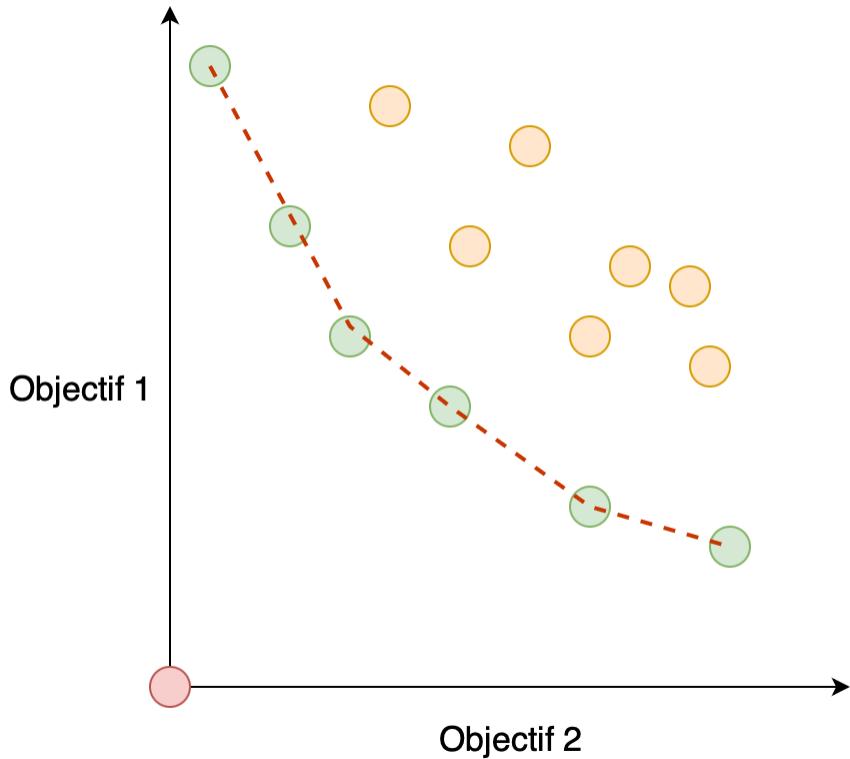


FIGURE 4: Front de Pareto pour deux objectifs à minimiser. Les points non dominés (vert) forment le front (ligne pointillée), tandis que les points dominés (orange) se situent dans la région supérieure. L'origine représente le point idéal (utopie) où il n'y a pas de compromis. Cette figure illustre que se déplacer le long du front améliore un objectif au prix de la dégradation de l'autre.

5.1.1 Méthodes de sélection d'un compromis sur le front

Une optimisation multiobjectifs peut produire un nombre élevé de solutions dites optimales, qui sont de bonnes solutions pour chaque objectif, ce qui peut rendre la prise de décision difficile pour un praticien. Différents articles ont proposé plusieurs méthodes pour extraire un seul compromis pertinent :

- **Approches multi-critères** : [Wang et Rangaiah](#) ont recensé et implémenté dix techniques (TOPSIS, LINMAP, VIKOR, GRA, SAW, MEW, ELECTRE II/III, NFM, FUCA), puis comparé leur efficacité sur des problèmes mathématiques et d'ingénierie chimique.

- **Détection de knee-points** : ces méthodes visent à repérer les "genoux" du front, où une amélioration marginale d'un objectif entraîne une forte dégradation d'un autre.
- **Scalarisation** : le problème multiobjectif est converti en un problème mono-objectif via une somme pondérée, ce qui permet de cibler directement le compromis souhaité. Bien que ce ne soit pas l'objet principal de cette méthode, [Gowda et al.](#) propose une métrique SAM qui pénalise les solutions augmentant la consommation d'énergie pour des gains de précision faibles, constituant ainsi une métrique pertinente pour décider en fonction d'un compromis exactitude-consommation d'énergie.

Cette liste n'est pas exhaustive, mais chacune de ces méthodes présente des avantages et des inconvénients. Pour plus de détails sur les dix techniques, voir : [Wang et Rangaiah..](#)

5.2 Approches évolutionnaires multiobjectif

5.2.1 Principes des algorithmes évolutionnaires

Les algorithmes évolutionnaires forment une classe d'algorithme s'inspirant du processus de sélection naturelle décrit par Charles Darwin. Bien que les implémentations des mécanismes d'évolutions varient beaucoup, l'idée fondamentale reste la même : parmi une population d'individus seuls, les mieux adaptés à un environnement aux ressources limitées survivent et se reproduisent. Chaque individu est évalué selon une fonction de *fitness*, qui mesure sa performance par rapport à un objectif fixé.

Ainsi ces algorithmes manipulent une population de solutions candidates (individus) qui évoluent collectivement. Cette population constitue l'ensemble des points explorés correspondant à l'espace de recherche. Le processus de sélection promeut les meilleures solutions grâce à des critères de qualités ou d'adaptabilité. Ensuite, les solutions sont croisées et générées avec des variations aléatoires pour agir comme la mutation génétique ([Afzal et Torkar](#)). Le principal objectif est de favoriser l'émergence de solution favorable à la résolution du problème.

Un individu correspond à une solution codée de manière adaptée au problème (par exemple un vecteur de réels, un vecteur binaire, etc.). Plusieurs paramètres principaux caractérisent le comportement de l'algorithme : la taille de la population, le nombre de descendants générés, les taux de croisement et de mutation. Des taux élevés favorisent l'exploration de régions non visitées de l'espace de recherche, tandis que des taux faibles renforcent l'exploitation de régions déjà connues.

Le croisement (*crossover*) est un mécanisme fondamental des AE, il fait évoluer les solutions collectivement en échangeant des informations entre les individus. Ce mécanisme combine l'information "génétique" de deux parents pour produire la génération d'individus suivante. Concrètement sont choisis un ou plusieurs points de découpe dans la représentation des individus. Ensuite, les segments correspondants entre les deux parents sont échangés, ce qui permet d'obtenir une nouvelle génération d'individus qui héritent de caractéristiques des deux donneurs. Grâce à ce mécanisme, l'algorithme peut combiner des éléments performants issus de solutions différentes. Enfin, cela favorise l'exploration de l'espace de recherche en maintenant de la diversité.

La mutation est un opérateur de variation qui modifie aléatoirement une ou plusieurs composantes d'un individu pour introduire de la nouveauté dans la population. En pratique, après le croisement, chaque enfant subit un test probabiliste ([Eiben et Smith](#)) : avec une faible probabilité (par exemple 1 % par gène) que la valeur d'un gène soit changée. Par exemple dans un codage binaire, on inverse un bit. Une autre méthode consiste en la mutation uniforme. La

valeur d'un gène sélectionné est remplacée par une nouvelle valeur tirée aléatoirement selon une distribution homogène, entre les bornes minimale et maximale définies par l'utilisateur pour ce gène ([Dagdia et Mirchev](#)). Grâce à ce mécanisme, la mutation empêche la population de se figer sur des régions locales de l'espace de recherche et favorise l'exploration de solutions inattendues, ce qui limite une convergence prématuée.

Il y a deux sélections à distinguer : la sélection des parents et la sélection des individus présents pour la génération suivante. La première vise à favoriser les individus de meilleure qualité pour leur permettre de devenir parents de la génération suivante (stratégie de remplacement). Un individu est parent s'il a été choisi pour créer une progéniture (*offspring*) qui introduit de la variation. Dans cette sélection, les individus de faible qualité ont peu de chances, cependant une probabilité positive leur est laissée afin d'éviter que la population reste bloquée dans un optimum local. Le deuxième type est la sélection de remplacement (ou des survivants), qui prend place après la création de la progéniture. Elle a pour objectif de conserver, pour la génération suivante, uniquement les individus en fonction de leur qualité. Plusieurs méthodes existent : notamment une méthode basée sur la fonction de fitness (*fitness-based*), qui consiste à classer parents et progéniture et à sélectionner les meilleurs éléments, ou une approche fondée sur l'âge (*age-biased*), qui régule la composition de la population en fonction de l'ancienneté des individus, certaines variantes ne choisissent que parmi la progéniture, d'autres combinent critères d'âge et de qualité ([Eiben et Smith](#)).

Globalement le scénario d'exécution d'un AE peut être décrit comme suit (fig. 5) :

Algorithme 1 : Algorithme évolutionnaire (générique)

Entrée : Paramètres : taille population, nombre de descendants

Sortie : Meilleur individu trouvé

- 1 Initialisation** Générer aléatoirement une population de n individus
 - 2 while** le critère d'arrêt n'est pas atteint **do**
 - 3 Évaluation** Calculer la fitness de chaque individu
 - 4 Sélection** Choisir n parents
 - 5 Variation** Appliquer croisement et mutation pour générer k descendants
 - 6 Remplacement** Former la nouvelle population
 - 7 Retourner** Le meilleur individu
-

Bien que ces principes soient efficaces pour l'optimisation mono-objectif, ils montrent deux limites majeures en contexte multiobjectifs. D'abord, l'absence d'une fonction de fitness unique rend impossible un classement total des solutions : plusieurs solutions peuvent représenter des compromis non dominés. Ensuite, les schémas de sélection classiques tendent à appauvrir la diversité si rien n'est fait pour conserver une couverture représentative du front de Pareto. NSGA-II adresse ces problèmes en combinant un tri non dominé pour gérer les compromis, une distance de foule (*crowding distance*) pour préserver la diversité locale, et un mécanisme d'élitisme via la sélection sur la population fusionnée.

5.2.2 NSGA-II

NSGA-II, introduit par [Deb et al.](#) (2002), est un algorithme évolutionnaire multiobjectifs (MOEA) conçu comme le successeur de NSGA. Son objectif est d'approximer le front de Pareto en corrigeant deux limitations majeures de NSGA : d'une part la complexité élevée du tri non dominé, qui était en $\mathcal{O}(MN^3)$ dans NSGA, et d'autre part l'absence d'élitisme, ce qui pouvait entraîner

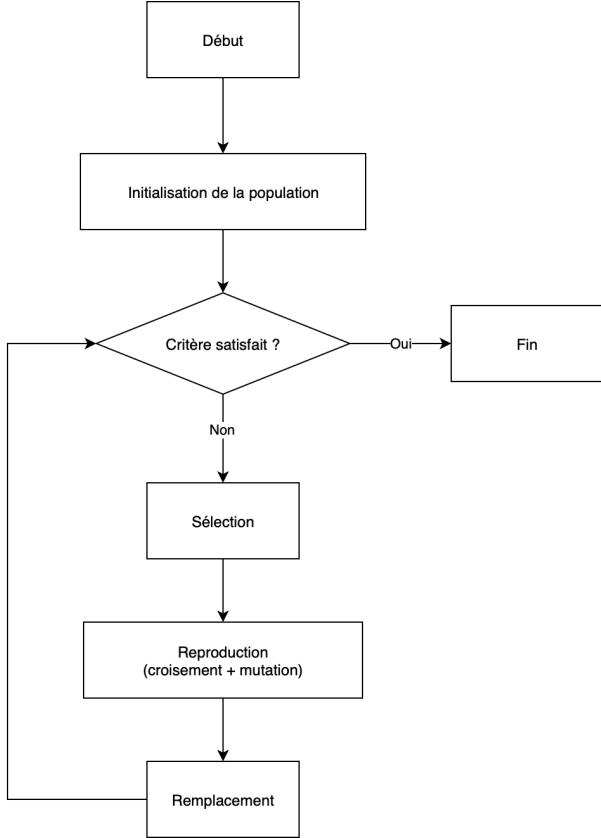


FIGURE 5: Diagramme illustratif du fonctionnement d'un algorithme évolutionnaire.

la perte de bonnes solutions entre générations. NSGA-II réduit la complexité du tri à $\mathcal{O}(MN^2)$ grâce à un *fast non-dominated sorting*, introduit un élitisme explicite par la fusion des parents et des descendants afin de conserver les meilleures solutions, et préserve la diversité au sein de chaque front en utilisant la *crowding distance* (distance de foule) dans l'opérateur de comparaison par foule (*crowded-comparison operator*), évitant ainsi la saturation de l'espace de recherche.

L'aspect le plus fondamental de NSGA-II est le tri par non-domination. Pour rappel, une solution a domine une solution b si a est au moins aussi bonne que b sur tous les objectifs, et strictement meilleure sur au moins un d'entre eux (sec 5.1).

Le tri par non-domination permet de regrouper la population en plusieurs fronts de solutions :

- F_1 : regroupe toutes les solutions qui ne sont dominées par aucune autre,
- F_2 : contient les solutions uniquement dominées par celles de F_1 ,
- F_3 : contient les solutions dominées par $F_1 \cup F_2$,
- etc.

Le but est de trier tous les individus de la population dans ces fronts successifs et d'assigner à chacun un "rang de domination" correspondant au front auquel il appartient. Une solution dans F_i se verra attribuer un rang i . Ce rang est un critère central, car il sert à comparer les individus lors de la sélection des parents et lors de la reconstruction de la population à chaque génération.

La version optimisée utilisée dans NSGA-II, appelée *Fast Non-Dominated Sorting*, permet d'effectuer ce tri plus efficacement en $\mathcal{O}(M \cdot N^2)$, où M est le nombre d'objectifs et N la taille de la population. Ce classement en fronts est également une étape préalable nécessaire au calcul de la crowding distance, celle-ci n'est définie qu'au sein d'un même front, et suppose donc que

le tri par non-domination ait déjà été effectué.

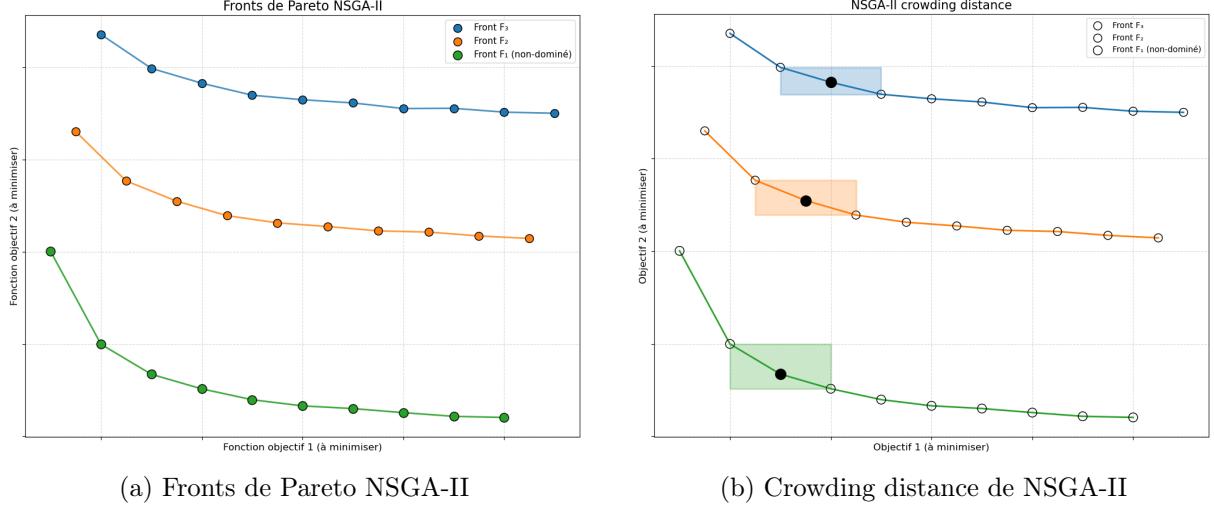


FIGURE 6: Illustration de deux aspects des résultats de NSGA-II : (a) répartition des solutions sur les fronts de Pareto et (b) mise en évidence de la crowding distance.

Dans NSGA-II, la crowding distance est une mesure de la densité locale des solutions autour d'un individu dans l'espace des objectifs. Elle sert à favoriser la diversité sur le front de Pareto en donnant la priorité aux individus les plus isolés lors de la sélection. Pour estimer la densité locale autour d'une solution i , on commence par trier les solutions de son front selon chaque objectif, indépendamment. Pour chaque objectif, on calcule la distance entre les deux solutions voisines les plus proches de i dans cet ordre trié. Ces distances sont ensuite additionnées pour obtenir la crowding distance de i , une mesure de la densité locale de cette solution. Cette valeur est interprétée comme une estimation de la taille d'un rectangle entourant i dans l'espace des objectifs. Plus la crowding distance est grande, plus la solution est isolée, ce qui indique une région peu dense. Inversement, une crowding distance faible indique que la solution se trouve dans une région densément peuplée.

La crowding distance est utilisée lors de la phase de sélection, à la fin de chaque génération. Après avoir combiné les parents et les enfants pour former une population intermédiaire. Cette fusion constitue le mécanisme d'élitisme, les solutions les plus performantes de l'ensemble combiné sont triées en fronts de non-domination (6a). Les fronts sont ensuite ajoutés un par un pour constituer la population suivante. Cela permet de garantir la préservation des meilleurs individus d'une génération à l'autre. Cependant, si un front dépasse la taille restante disponible, seules certaines solutions sont retenues. Dans ce cas, les individus sont triés selon leur crowding distance (6b), et les plus isolés, avec la plus grande distance, sont préférés afin de maintenir une bonne diversité au sein de la population.

Les fronts de domination et la crowding distance sont utilisés conjointement pour départager des individus. Appelé *crowded-comparison operator*, il guide la sélection à deux moments de l'algorithme : lors de la sélection des parents pour générer les descendants, et lors de la sélection environnementale pour construire la nouvelle population, afin de départager les individus appartenant au même front. Son but est de favoriser un front de Pareto à la fois de qualité et uniformément diversifié. Il privilégie d'abord les individus ayant le rang de domination le plus faible. Lorsque plusieurs individus sont sur un même front, il retient ceux situés dans des zones moins denses.

Ci dessous est décrit en pseudo-code l'exécution complète de l'algorithme NSGA-II :

Algorithme 2 : NSGA-II

Entrée : Paramètres : N (taille population), critère d'arrêt
Sortie : Approximation du front de Pareto

1 Initialisation Initialiser aléatoirement une population P_0 de taille N

2 Évaluation Évaluer tous les individus sur les fonctions objectifs

3 while le critère d'arrêt n'est pas respecté **do**

4 Évaluation Calculer la fitness de chaque individu

5 Sélection Sélectionner par tournoi binaire une population de parents à partir de P_t grâce au crowded-comparison operator

6 Variation Appliquer un crossover et mutation pour produire une descendance Q_t de taille N .

7 Fusionner Former la population combinée $R_t = P_t \cup Q_t$, taille $2N$

8 Trier Appliquer le fast non-dominated sorting sur R_t pour obtenir les fronts F_1, F_2, F_3, \dots

9 Calculer Calculer la crowding distance pour chaque front

10 Construire Construire la nouvelle population P_{t+1} en ajoutant les fronts successifs jusqu'à atteindre N si un front ne tient pas entièrement, trier ce front par crowding distance décroissante et remplir les places restantes.

11 Retourner Approximation du front de Pareto : F_1

L'algorithme n'est pas réimplémenté dans ce travail afin de se focaliser sur l'optimisation des hyperparamètres et l'analyse des compromis exactitude-efficacité.

5.2.3 Algorithmes évolutionnaires multiobjectifs alternatifs

Même si NSGA-II sert de base à l'optimisation des hyperparamètres de ce travail, il existe d'autres familles d'algorithmes évolutionnaire multiobjectifs. Cette section les présente brièvement.

Algorithmes basés sur la dominance Ces méthodes utilisent directement le principe de non domination pour guider la sélection. Parmi elles, SPEA2 ([Zitzler et al.](#)) performant avec des problèmes à plus de deux objectifs, il améliore aussi la sélection basé sur la densité par rapport à la première version SPEA. PAES ([Knowles et Corne](#)) repose sur une stratégie d'évolution $(1+1)$ ⁴ combinée à une recherche locale, et utilise une archive pour estimer le rang de domination des solutions explorées. PESA-II ([Corne et al.](#)) introduit une sélection régionale dans l'espace d'objectif pour favoriser un front de Pareto uniforme.

Algorithmes Indicator-Based Contrairement aux méthodes fondées uniquement sur la dominance, des algorithmes comme IBEA ([Zitzler et Künzli](#)) intègrent un indicateur de performance dans la sélection. Cependant, selon l'indicateur choisi, cela peut réduire l'efficacité du MOEA sans nécessairement améliorer la convergence ou la répartition des solutions ([Wang et al.](#)), ce qui rend cette approche particulièrement sensible au choix de l'indicateur.

⁴. Une stratégie évolutionnaire $(1+1)$ désigne un algorithme où une seule solution parentale génère une seule descendance à chaque itération, et seule la meilleure des deux est conservée.

Approche par décomposition MOEA/D décompose un problème multiobjectifs en plusieurs sous-problèmes scalaires optimisés de manière coopérative. Chaque sous-problème ne s'appuie que sur l'information échangée avec ses voisins pour guider sa recherche (Zhang et Li). Grâce à cette décomposition, et au fait que les sous-problèmes peuvent être traités en parallèle, l'approche se prête bien à une implémentation parallèle. Cependant, la qualité de l'approximation du front dépend fortement de la distribution initiale des vecteurs de pondération choisis (Wang et al.).

5.3 Optimisation multiobjectif avec Optuna

Optuna est un framework open source d'optimisation développé par Akiba et al.. Il est dédié à faciliter la tâche complexe et fastidieuse de l'optimisation des hyperparamètres. Cela est organisée sous forme d'une étude (*study*), constituée d'itérations appelées essais (*trials*), dans lesquelles le framework évalue différentes configurations d'hyperparamètres automatiquement. Chaque nouvelle configuration est proposée par un algorithme de recherche (*sampler*) mono ou multiobjectif, ici NSGA-II. Les résultats obtenus sont enregistrés dans un espace de stockage (*storage*) et utilisés pour orienter la recherche vers des régions intéressantes dans l'espace de recherche.

Le framework se démarque par sa flexibilité dans la définition de l'espace de recherche. Il permet de spécifier des hyperparamètres continus, entiers, discrets ou catégoriels avec des distributions adaptées aux différents cas. Par exemple, la distribution log-uniforme est adaptée au réglage du learning rate car elle facilite la prise en compte de variations importantes d'ordre de grandeur. Dans le cadre de l'optimisation multiobjectif, Optuna gère nativement la génération de solutions non dominées. Elle intègre aussi des outils de visualisation, comme la représentation du front de Pareto ou l'analyse de l'importance relative des hyperparamètres (fANOVA).

Dans ce travail, Optuna est employé pour son implémentation de NSGA-II et pour générer une approximation du front de Pareto. Ses outils intégrés d'analyse et de visualisation sont utilisés pour interpréter les compromis identifiés.

Listing 1: Exemple minimal simplifié d'utilisation d'Optuna avec NSGA-II pour optimiser deux objectifs

```
import optuna

# Fonction objectif multiobjectif
def objective(trial):
    lr = trial.suggest_loguniform("lr_backbone", 1e-5, 1e-3, log=True)
    dropout = trial.suggest_uniform("dropout", 0.0, 0.5)
    train(model)
    accuracy = eval(model, validation_split)
    energy = logger.energy()
    return accuracy, energy # Résultats de l'essai

# Création et exécution de l'étude
study = optuna.create_study(
    directions=["maximize", "minimize"], # exactitude, énergie
    sampler=optuna.samplers.NSGAISampler()
)
study.optimize(objective, n_trials=50)

# Affichage des solutions évaluées présentent du front de Pareto
print("Nombre de solutions optimales :", study.best_trials)
```

Le code Python ci-dessus illustre plus en détail comment Optuna fonctionne et sera utilisé dans la méthodologie expérimentale. La fonction `objective`, appelée par l'étude pour effectuer une évaluation, retourne les résultats qui seront exploités par le sampler afin de guider l'optimisation. La variable `trial`, transmise en argument, contient les métadonnées de l'étude en cours, notamment les hyperparamètres déjà testés. Grâce à `trial.suggest`, de nouvelles valeurs sont attribuées après analyse de l'historique des attributions précédentes. Le modèle est ensuite évalué avec cette configuration et les performances obtenues sont renvoyées. Cela se répète jusqu'à atteindre le nombre d'itérations défini au préalable. De plus, l'utilisation d'un *validation split* est recommandée afin d'éviter que le sampler ne s'adapte au jeu de test cela conduirait à une estimation biaisée des performances réelles. C'est pour cette raison qu'il faut lorsque l'étude est terminée vérifier les configurations jugées pertinentes sur le test set afin de mesurer leur véritable performance.

Des alternatives à Optuna existent, telles que SMAC ([Hutter et al.](#)) ou Hyperopt ([Bergstra et al.](#)) qui reposent sur d'autres stratégies d'optimisation, comme l'optimisation bayésienne ou le TPE (*Tree-Structured Parzen Estimator*).

6 Méthodologie expérimentale

L'objectif est de concrètement ancrer les points théoriques cités tout au long du mémoire en un exemple pratique et pertinent, permettant de mieux relier la théorie et la mise en oeuvre. L'expérimentation vise à illustrer de manière claire les concepts d'optimisation multiobjectif identifiés dans l'état de l'art. L'expérimentation est construite en trois phases d'abord l'approximation du front de Pareto. Ensuite, à partir de ce front, 3 solutions sont retenues et entraînées à "haute fidélité" pour vérifier la qualité et la cohérence du front obtenu. L'importance de chaque hyperparamètre est évaluée en fonction de son objectif afin de mieux comprendre leur rôle dans le compromis entre exactitude et efficacité.

6.1 Environnement matériel et logiciel

Les détails matériels et logiciels utilisés pour toutes les expériences sont présentés afin de permettre leur reproduction.

Matériel Les caractéristiques matérielles utilisée sont les suivantes :

- **Machine** : Serveur distant (Runpod).
- **Processeur (CPU)** : 8 cœurs (modèle non précisé par Runpod)
- **GPU** : NVIDIA RTX ADA 4000 (20 Go VRAM)
- **Mémoire vive (RAM)** : 47 Go
- **Stockage** : NVMe SSD

Logiciel L'environnement logiciel est encapsulé dans l'image Docker utilisée pour toutes les expériences : `runpod/pytorch:2.8.0-py3.11-cuda12.8.1-cudnn-devel-ubuntu22.04`, qui fournit un système basé sur **Ubuntu 22.04**, **Python 3.11**, **PyTorch 2.8.0** ainsi que **CUDA 12.8.1** et les bibliothèques GPU associées.

Reproductibilité

- Le code est versionné avec `git` à l'adresse : <https://github.com/mathieu-plapied/M60-Experiment>. Le dépôt comprend un fichier `README` détaillant l'ensemble des instructions nécessaires à la réexécution du code.
- Les bibliothèques python utilisées sont fournies dans le fichier `requierement.txt`.
- Les résultats de l'optimisation Optuna NSGA-II sont sauvegardés dans un fichier `optuna.db` et peuvent être réutilisés pour une analyse ultérieure.
- L'ensemble des sources de non-déterministe sont contrôlées via fixation des seed sur `numpy`, `torch`, `random`, etc. sont fixé à la valeur 42 (`SEED=42`) pour l'ensemble du projet. Les autres paramètres qui influencent le déterminisme sont adaptés (désactivation de `cudnn.benchmark`, `torch.backends.cudnn.deterministic`, etc.).

6.2 Modèle, jeu de données, optimiseur et métriques

Modèle

Le modèle retenu pour cette expérimentation est **EfficientNetV2-Small**, préentraîné sur ImageNet-1k, afin de tirer parti du transfer learning et d'accélérer la phase initiale d'entraînement. La

version "Small" présente une complexité modérée tout en restant adaptée aux tâches de type *fine-grained*. Son architecture est décrite bloc par bloc dans la table 4. Ce choix est motivé par une bonne efficacité computationnelle du modèle, en plus des bonnes performances sur une tâche complexe qu'est ImageNet ([Tan et Le](#)). De plus, sa version préentraînée est directement disponible dans le framework PyTorch ce qui facilite l'intégration. Enfin, des contraintes matérielles ont également orienté ce choix, car les versions M ou L auraient exigé des ressources matérielles plus importantes.

Bloc	Opérateur	Kernel	SE	Expansion	Stride	Pad	Canaux	Répétitions
1	Conv 3×3	—	—	—	2	1	24	1
2	Fused-MBConv1	3×3	—	1	1	1	24	2
3	Fused-MBConv4	3×3	—	4	2	1	48	4
4	Fused-MBConv4	3×3	—	4	2	1	64	4
5	MBConv4	3×3	0,25	4	2	1	128	6
6	MBConv6	3×3	0,25	6	1	1	160	9
7	MBConv6	3×3	0,25	6	2	1	256	15
8	Conv 1×1 + Pooling	—	—	—	—	0	1280	1

TABLE 4: Composition par bloc d'EfficientNet V2-S (small).

Jeu de données et prétraitement

Le jeu de données **Stanford Dogs** a été retenu pour mener les expérimentations, car il constitue un benchmark de classification d'images de type "fine-grained" ([Khosla et al.](#)). Ce choix en fait un cas d'étude pertinent pour évaluer le compromis exactitude–efficacité dans un contexte réaliste. Il comporte 120 classes de races de chiens et un total de 20 580 images. Elles sont réparties en un train set de 12 000 exemples et un test set de 8 580 exemples. Cela offre une diversité suffisante et une taille gérable pour que l'expérimentation puisse être reproduite et rester intéressante. Ce jeu de données a été préféré à d'autres comme *Oxford-IIIT Pet* en raison de son nombre de classes plus élevé et de sa complexité, mieux adaptée à EfficientNetV2-S.

De plus EfficientNetV2-S a été préentraîné sur ImageNet. Ce jeu de données de classification comprend 1 000 classes, dont 118 races de chiens. Stanford Dogs étant un sous-ensemble de ce corpus, il présente une similarité favorable au transfert learning. Cette proximité réduit l'écart de domaine entre préentraînements et fine-tuning, ce qui, comme l'indiquent [Liu et al.](#), tend à améliorer les performances par rapport à un transfert inter-domaine. Les représentations visuelles acquises constituent ainsi une base pertinente pour fine-tuner le modèle sur les spécificités des races de chiens présentes dans Stanford Dogs.

Afin de prévenir le surapprentissage et d'améliorer la généralisation, les images d'entraînement sont soumises à une augmentation aléatoire (RandomResizedCrop(224), RandomHorizontalFlip), puis converties en tenseur et normalisées selon les statistiques ImageNet.

Pour l'évaluation, les images sont redimensionnées à 256px, recadrées aux centres, transformées en tenseur et subissent la même normalisation basée sur ImageNet que pour les données d'entraînement.

Optimiseur

Optuna a été retenu pour l'optimisation multiobjectif des hyperparamètres. D'abord, il intègre nativement NSGA-II, permettant de générer directement un ensemble de solutions non dominées. Ensuite, la direction des objectifs (maximiser ou minimiser) se spécifie explicitement sans modification nécessaire de la fonction objectif. Cela réduit la complexité de mise en oeuvre et le risque d'erreurs. Les essais et leurs métriques sont enregistrés dans un stockage dédié, ce qui facilite une analyse ultérieur et la reproductibilité. Enfin, ma familiarité préalable avec l'outil et sa mise en place rapide dans l'environnement expérimental ont permis de limiter le temps et les efforts nécessaires à son adoption.

Métriques

Métrique énergétique J/image a été retenu plutôt que J/FLOP. Sur une même machine, avec la même précision et un GPU correctement saturé, l'énergie par opération (J/FLOP) reste globalement stable d'un essai à l'autre. Elle renseigne surtout sur l'efficacité matérielle. Tandis que J/image reflète le coût énergétique de complet de l'entraînement, sensible aux hyperparamètres qui modifient le débit d'images et la pression sur la mémoire. Cette métrique est donc plus alignée avec l'objectif de compromis exactitude vs. efficacité énergétique.

Métriques d'exactitude Pendant l'optimisation multiobjectif, l'exactitude est utilisée comme objectif de performance sur l'ensemble de validation (en parallèle de la métrique énergétique). Ce choix privilégie une mesure simple et facilement comparable d'une configuration à l'autre. Lors de l'analyse finale des trois configurations retenues sur le test set les métriques F1-macro et Top-3 sont utilisées pour compléter le compromis obtenu.



English Springer



Irish Setter



Irish Wolfhound



Lakeland Terrier



Norwich Terrier



Clumber



Collie



Ibizan Hound



Dhole



Labrador Retriever



Chihuahua



Beagle

FIGURE 7: Échantillon d'images présentes dans le jeu de données Stanford Dogs.

6.3 Optimisation des hyperparamètres

L'objectif de cette étape est de rechercher automatiquement des configurations d'hyperparamètres qui offre un compromis favorable entre exactitude et efficacité énergétique grâce à l'approximation du front de Pareto. Cela est formulé comme un problème d'optimisations multiobjectif. Optuna ([Akiba et al.](#)) est utilisé comme framework avec l'optimiseur `NSGAIISampler` pour guider l'optimisation des deux objectifs : maximiser l'exactitude (eq. 4.1) et minimiser l'énergie par image tel que :

$$f_1 = acc_{top1}$$

$$f_2 = \frac{E_{\text{totale}}}{\text{nombre total d'images traités}}$$

où :

- f_1 l'exactitude du modèle sur le validation split.
- f_2 est l'énergie moyenne dépensée en joules pour traiter une image pendant l'entraînement.

La consommation énergétique du GPU est relevée en continu toutes les 200 ms à l'aide du paquet `pynvml` compatible avec les GPU Nvidia, qui fournit la puissance instantanée en watts. Par manque d'accès aux informations de consommation globale de la machine, l'optimisation se concentre exclusivement sur le GPU. La mesure démarre au début de la première époque de chaque essai et s'interrompt à la fin de la dernière.

À chaque intervalle de 200 ms, la puissance instantanée P_i (en watts) est relevée. L'énergie totale consommée sur la durée T est estimée par une somme discrète :

$$E_{\text{totale}} = \sum_{i=1}^N P_i \times \Delta t, \quad \Delta t = 0,2 \text{ s}, \quad N = \frac{T}{\Delta t}$$

Chaque terme $P_i \times \Delta t$ correspond à l'énergie (en joules) consommée pendant l'intervalle Δt .

La métrique énergie par image est alors calculée en normalisant l'énergie totale par le nombre d'images traitées (entraînement + validation) et par le nombre d'epochs :

$$E_{\text{image}} = \frac{E_{\text{totale}}}{(|train| + |validation|) \times N_{\text{epochs}}},$$

où :

- E_{totale} est l'énergie cumulative mesurée (en joules),
- $|train|$ et $|validation|$ sont respectivement les tailles des jeux d'entraînement et de validation,
- N_{epochs} est le nombre d'epochs (ici, 8).

Le jeu de données d'entraînement est divisé en un nouveau training set et un validation set selon un split stratifié (20% du training set complet) afin de préserver la répartition des classes lors de l'évaluation. Le test set n'est pas utiliser et le sera uniquement lors de l'évaluation finale après l'entraînement complet d'un modèle. Le validation split est essentiel pour évaluer objectivement la qualité d'une configuration d'hyperparamètres, il empêche Optuna de favoriser les hyperparamètres qui sont uniquement adaptés au training set, au détriment de la généralisation.

Par contrainte de ressource, les essais (trials) d'optimisation sont effectués sur 40% du

jeu d'entraînement stratifié. Cela permet d'explorer rapidement l'espace d'hyperparamètres tout en gardant une représentativité suffisante pour obtenir des configurations prometteuses. Parmi les meilleures configurations présentes sur le front de Pareto, trois seront ensuite réentraînées complètement avec un nombre plus élevé d'epochs.

Les résultats de l'étude Optuna sont sauvegardés dans le fichier `optuna_study.db`. Ce fichier permet de charger facilement les solutions obtenues lors de l'étude des meilleurs hyperparamètres sans devoir relancer une nouvelle optimisation.

6.3.1 Espace de recherche : hyperparamètres

Parmi les hyperparamètres plusieurs sont fixées a priori pour réduire la dimension du problème et s'appuyer sur des composantes bien établies. L'optimiseur utilisé est `AdamW`, choisi pour sa dissociation entre le weight decay et la mise à jour des gradients, ce qui permet de les régler le learning rate et la weight decay. Pour limiter le surapprentissage et favoriser une meilleure généralisation, un *label smoothing* est appliqué lors de l'entraînement. Ces choix restent constants pendant l'optimisation des autres hyperparamètres.

L'espace de recherche contient les hyperparamètres suivants :

Hyperparamètre	Description	Domaine	Échantillonnage
<code>lr_backbone</code>	Taux d'apprentissage du backbone pré-entraîné	$[1 \times 10^{-5}, 1 \times 10^{-3}]$	Log-uniforme
<code>lr_classifier</code>	Taux d'apprentissage du classificateur	$[1 \times 10^{-4}, 1 \times 10^{-2}]$	Log-uniforme
<code>dropout</code>	Taux de dropout appliquée en sortie de blocs	$[0, 0.5]$	Uniforme
<code>weight_decay</code>	Coefficient de régularisation L2	$[1 \times 10^{-6}, 1 \times 10^{-2}]$	Log-uniforme
<code>n_blocks</code>	Nombre de blocs du backbone dégelés	$\{1, 2, \dots, 8\}$	Entier uniforme
<code>batch_size</code>	Taille des batchs utilisées pendant l'entraînement	$\{32, 64, 128, 192\}$	Catégoriel

Les intervalles indiqués correspondent aux domaines explorés par Optuna. Ils ont été choisis pour couvrir des ordres de grandeur pertinents tout en restant raisonnables vis-à-vis du budget de calcul. L'emploi de taux d'apprentissage distincts pour le backbone et pour la tête de classification repose sur le *discriminative fine-tuning* ([Howard et Ruder](#)) où chaque partie du modèle a son propre taux. Le classifieur utilise un taux plus élevé que le backbone, plutôt qu'un seul taux global, ce qui permet d'adapter fortement les couches proches de la sortie sans dégrader les représentations plus générales apprises dans les couches inférieures.

Le nombre de blocs (`n_blocks`) correspond à l'architecture du modèle EfficientNetV2-S présenté dans la table 4. Le modèle est divisé en 8 blocs distincts, lors de l'optimisation l'entièreté

du modèle peut être dégelée c'est-à-dire les 8 blocs ce qui implique un réentrainement complet. Sinon l'entraînement est partiel et les paramètres des blocs inférieurs ne sont pas entraînables. De plus si le backbone entier du modèle est gelé, il s'agit d'un transfer learning par extraction de caractéristiques.

La batch size est limitée à 192 par la mémoire VRAM du GPU qui risque de lancer une erreur si sa valeur devient plus élevée à cause d'une allocation dépassée.

6.3.2 Configuration NSGA-II

Le choix des hyperparamètres de NSGA-II est essentiel, car il conditionne directement la qualité de l'approximation du front de Pareto. Une population trop réduite compromet la diversité des solutions, alors qu'une population trop importante augmente fortement le coût computationnel et ralentit la convergence de l'algorithme. L'article original présentant NSGA-II utilise une population de 100 individus sur 250 générations, soit 25 000 évaluations. Une telle échelle n'est pas envisageable pour mon optimisation, car chaque individu correspond à un entraînement de 8 epochs. Dans l'expérimentation de ce mémoire, j'ai donc retenu une population de 30 individus évoluant sur 5 générations, ce qui représente 150 évaluations. Cette configuration respecte les recommandations d'Optuna pour NSGA-II qui requiert entre 100 et 1000 évaluations.

6.3.3 Importance des hyperparamètres par objectif

Le framework Optuna permet le calcul, via fANOVA de l'importance des hyperparamètres pour chaque objectif à partir des essais. Cela facilite l'interprétation du front de Pareto pour identifier les hyperparamètres les plus influents pour chaque objectif.

6.4 Entrainement haute fidélité

Le front de Pareto a été approximé avec un budget réduit, en utilisant seulement 40% du train set et en effectuant seulement 8 epochs par essai. Pour vérifier si ce front permet bien de guider un compromis entre exactitude et efficacité, trois configurations d'hyperparamètres seront sélectionnées pour réentraîner EfficientNetV2-S, cette fois-ci sur l'ensemble du *train set* et pendant 30 epochs. Le reste du protocole d'entraînement reste identique à celui utilisé lors de l'optimisation des hyperparamètres. Enfin, les trois solutions seront évaluées sur le test set avec l'exactitude top-1, la F1-macro et l'exactitude top-3, ainsi que la mesure de l'efficacité énergétique.

Les trois configurations choisies représentent des compromis distincts : (i) exactitude la plus élevée, (ii) genou du front et (iii) efficacité maximale. Ce trio illustre concrètement les options sans multiplier inutilement les expériences.

Le choix de 30 epochs est dû aux essais menés sur 40% du train set lors du développement. Ils montrent déjà un début de plateau après 8 epochs. Cependant, ce plateau est obtenu sur un ensemble réduit de données et ne garantit pas la convergence lorsqu'on passe à l'ensemble du train set. Pour la phase haute fidélité, le nombre d'epochs est fixé 30. Cela assure la convergence même pour les configurations apprenant plus lentement et une marge de sécurité pour capturer les gains résiduels souvent nécessaires pour départager finement les solutions.

Le genou est détecté avec le paquet `kneed` (algorithme de [Satopaa et al.](#)), qui recherche la plus forte concavité de la courbe.

L'objectif est de vérifier la fidélité du front, l'ordre relatif et les écarts entre solutions doivent rester cohérents quand le budget augmente. Si les configurations sélectionnés demeurent non dominé cela confirmera que le front guide effectivement le choix précision–efficacité énergétique.

7 Résultats : analyse et discussion

Cette section présente les résultats obtenus lors de l'expérimentation. Les données issues de l'optimisation multiobjectif sont d'abord examinées à travers le front de Pareto, avant d'évaluer les performances des configurations sélectionnées lors de l'entraînement haute fidélité. Enfin, l'importance relative des hyperparamètres est étudiée afin d'identifier ceux qui influencent le plus la position des solutions sur le front.

7.1 Front de Pareto

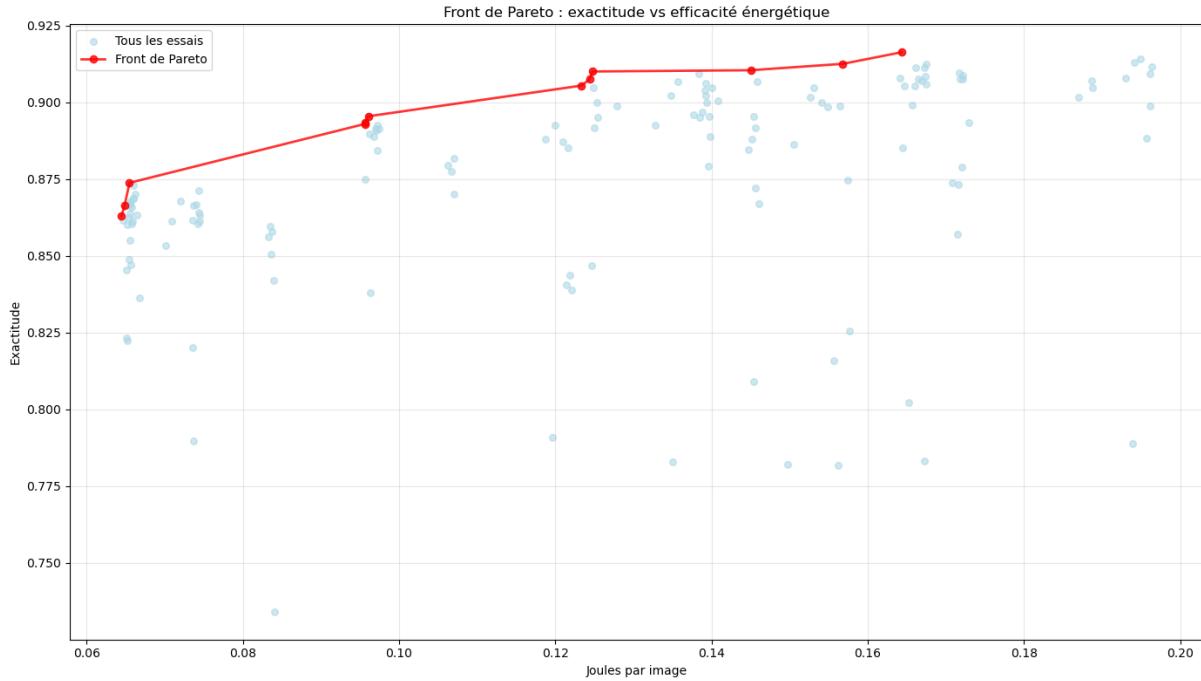


FIGURE 8: Front de Pareto obtenu après l'optimisation multiobjectif réalisée sur 40% du training set et 8 epochs par configuration. Chaque point bleu représente une configuration d'hyperparamètres évaluées. L'abscisse indique l'efficacité énergétique (J/image) et l'ordonnée l'exactitude top-1. Chaque point rouge correspond donc au front de Pareto et représente le meilleure compromis entre exactitude et efficacité énergétique.

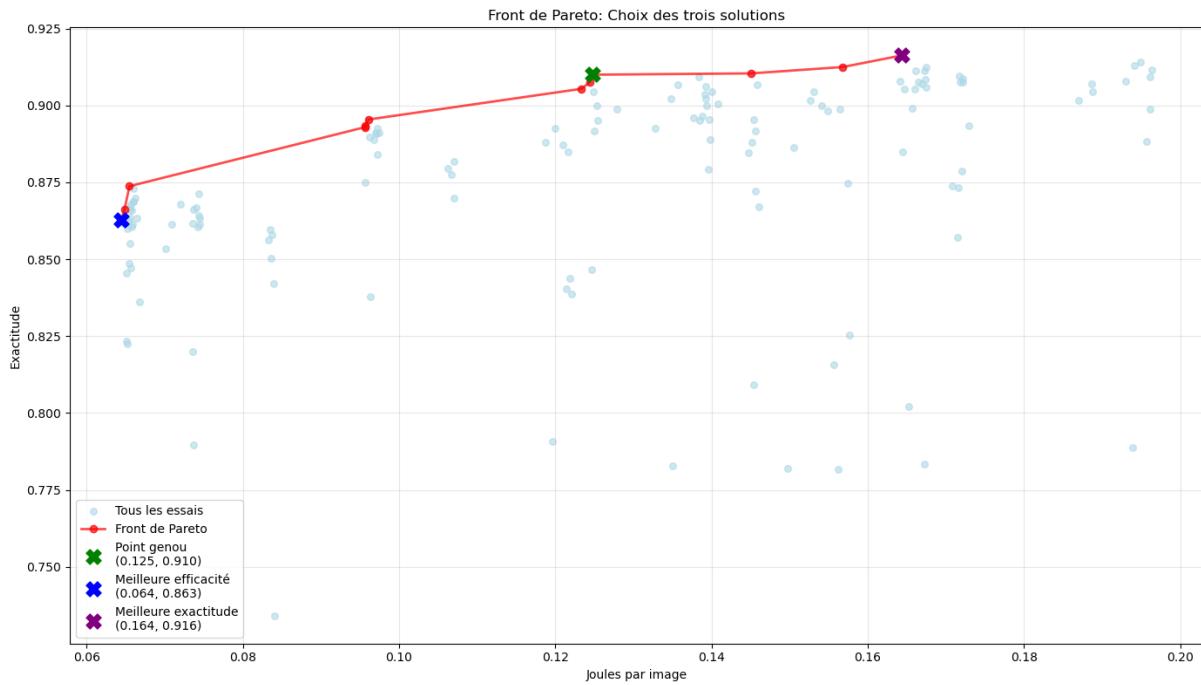


FIGURE 9: Les trois configurations choisies sur le front de Pareto. La croix bleue représente la solution avec la meilleure efficacité. La croix verte représente le point de genou obtenu. Enfin, la croix mauve représente la configuration qui a obtenu la meilleure exactitude.

La figure 8 met en évidence un front de Pareto concave, où l'on observe que les gains en exactitude deviennent marginaux à mesure que l'énergie par image augmente. Ce phénomène de rendement décroissant s'explique par le fait qu'EfficientNetV2-S est préentraîné sur des données déjà proches de celles nécessaires pour évaluer correctement le split de validation. Le modèle atteint donc dès le départ une exactitude relativement élevée, rendant chaque amélioration supplémentaire plus coûteuse en énergie. On observe ainsi qu'entre 0,06 J/image et 0,10 J/image, chaque augmentation de 0,04 J permet de gagner environ 2,5 points de pourcentage d'exactitude. En revanche, au-delà de 0,12 J/image, ce même gain énergétique n'apporte plus qu'environ 0,6 point de pourcentage.

Le premier changement net de la pente se trouve au point de genou à 91,0 % acc et 0,125 J/image. Ce point est intéressant car il offre 98 % de la précision maximum tout en économisant 31,6 % d'énergie par rapport à la solution la plus énergievore. Cela en fait un bon compromis sur le front de Pareto.

Parmi les 150 configurations explorées, seulement 12 se situent sur le front de Pareto, soit 8 %. En d'autres termes, plus de neuf solutions sur dix présentent à la fois une exactitude inférieure ou égale et une consommation énergétique supérieure ou égale à une autre solution. Cette observation souligne qu'un choix aléatoire aurait 92 % de chances de conduire à un modèle objectivement sous-optimal. Cela justifie pleinement le recours à une recherche multiobjectif structurée. La rareté des solutions non dominées montre que l'optimisation multiobjectif apporte un gain réel en ciblant les zones les plus prometteuses du front, tout en évitant de consacrer des ressources de calcul à des combinaisons vouées à être sous-optimales.

7.2 Entrainement haute fidélité

Cette section évalue si le front de Pareto obtenu lors de l'optimisation multiobjectif en configuration basse fidélité conserve sa pertinence pour guider le choix d'un compromis exactitude-efficacité énergétique après un entraînement haute fidélité.

Après le réentraînement complet sur 30 epochs et 100 % du train set, la configuration "exactitude max" est désormais dominée par le point de genou. Elle présente une précision très proche (-0,41 pp), mais consomme 37 % d'énergie par image en plus (Table 5). Le point de genou dépasse également cette configuration sur l'ensemble des autres métriques.

TABLE 5: Comparaison des performances selon la configuration basée sur les test set

Configuration	Exactitude (%)	Top-3 Accuracy (%)	F1-macro (%)	J/image
Efficacité max	88.00	97.68	87.32	0.06493
Point de genou	91.54	98.78	91.11	0.14385
Exactitude max	91.13	98.65	90.62	0.19650

Le fait que la configuration "exactitude maximale", initialement non dominée, devienne dominée par le point de genou s'explique par la dynamique d'apprentissage des modèles. On observe que la configuration "exactitude maximale" avait déjà atteint un plateau à 91,6 % sur le split de validation, tandis que celui correspondant au point de genou disposait encore d'une marge de progression en passant de 8 à 30 epochs. Ce résultat valide l'usage du front comme outil de décision, car le genou ici fournit ici le meilleur compromis exactitude-énergie dans des conditions réalistes et confirme que l'approximation basse fidélité est informative pour trier les candidats. Cependant une phase de validation haute fidélité est nécessaire avant un choix final.

Ces résultats confirment que le front obtenu par optimisation multiobjectif guide le choix d'un compromis exactitude-efficacité énergétique. Ils confirment aussi la pertinence du point de genou comme compromis recommandable, il améliore l'exactitude maximale tout en économisant environ 0,05 J par image.

Afin de renforcer la légitimité des conclusions, il aurait été pertinent de répéter chaque entraînement avec plusieurs valeurs de `seed` aléatoire, puis de reporter la moyenne et l'écart-type des mesures. Cela permettrait de réduire l'impact d'une variation aléatoire liée à l'initialisation de poids ou à l'ordre d'exécution des batchs. Pour des raisons de temps et de ressources, cette vérification n'a pas été réalisée dans le cadre de ce travail. Elle constitue toutefois une piste d'amélioration pour des travaux futurs.

7.3 Importance des hyperparamètres

La figure 10 montre que le nombre de blocs dégelés (`n_blocks`) explique environ 93 % de la variance de l'énergie par image (via fANOVA). C'est donc le levier prioritaire pour sur l'axe de l'efficacité (J/image). La taille de lot (`batch_size`) a un effet plus modéré. Les autres hyperparamètres ont une influence négligeable sur l'efficacité énergétique. Concrètement, `n_blocks` et, dans une moindre mesure, `batch_size` déplacent la solution le long de l'axe horizontal (J/image) du front de Pareto. Ce constat est cohérent avec la théorie : l'énergie consommée est principalement déterminée par la structure du réseau. Le nombre de couches dégelées impacte le nombre

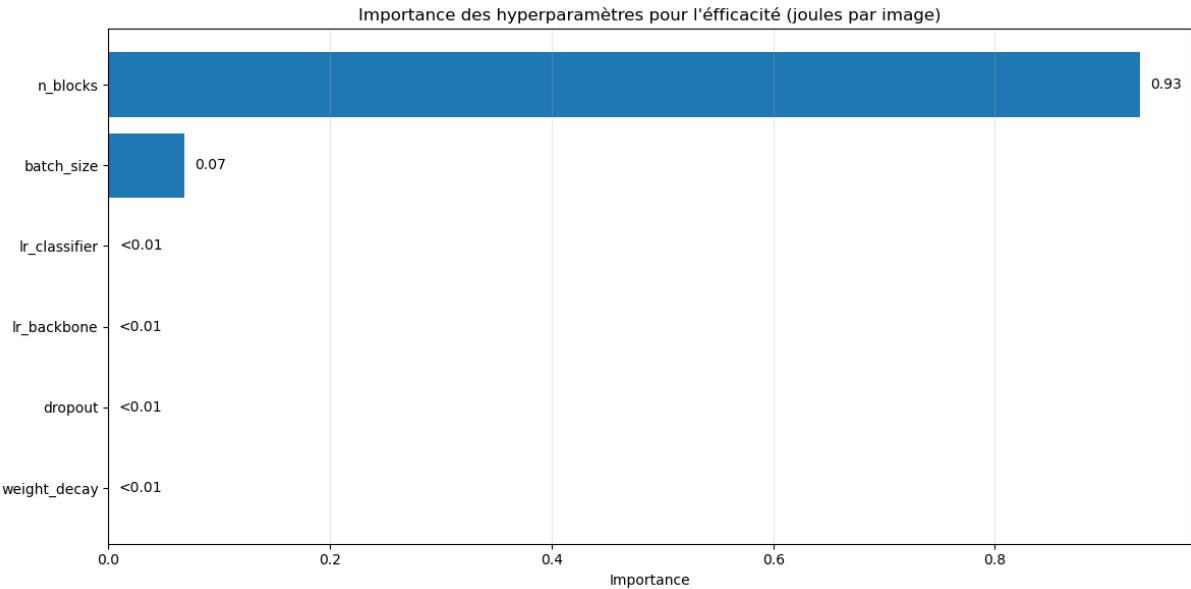


FIGURE 10: Importance des hyperparamètres par rapport à l'efficacité en %.

d'opérations nécessaires pour traiter une image. Ainsi, pour améliorer l'efficacité énergétique le long de l'axe correspondant sur le front de Pareto, il convient en priorité d'ajuster le paramètre `n_blocks`.

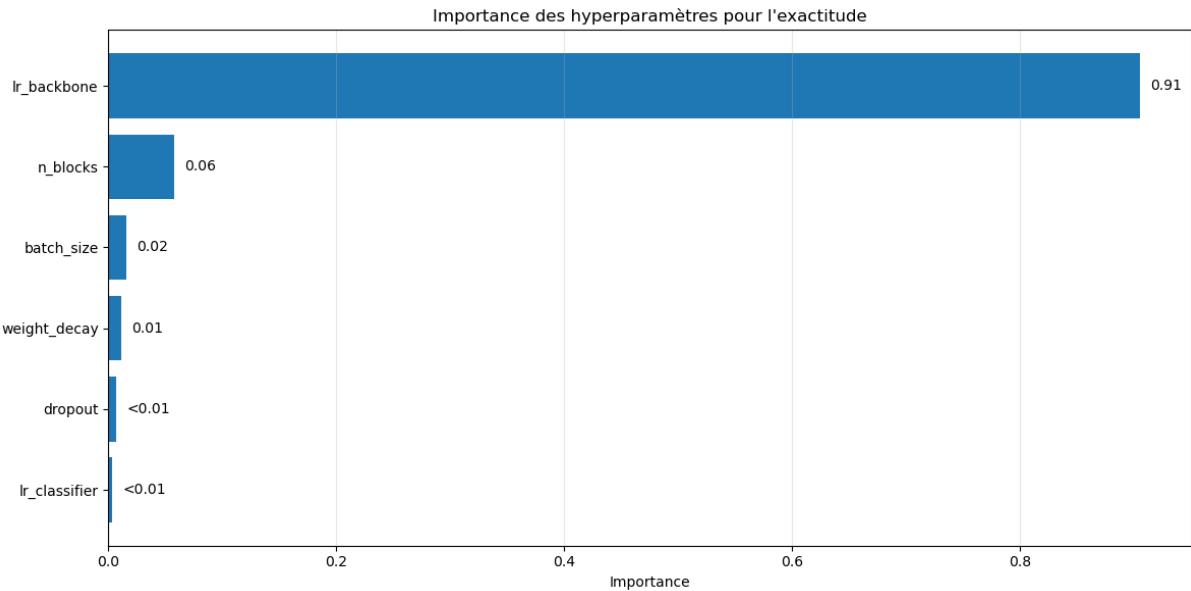


FIGURE 11: Importance des hyperparamètres par rapport à l'exactitude en %.

À l'inverse, l'exactitude est principalement gouvernée par le taux d'apprentissage appliqué au backbone (`lr_backbone`), comme le montre la figure 11. L'effet de `n_blocks` sur l'exactitude reste secondaire, et celui de `batch_size` marginal dans notre protocole. Il faut rappeler que fANOVA mesure des importances marginales et n'indique pas le sens des effets. En pratique, `lr_backbone` pilote surtout la position verticale (exactitude) des solutions sur le front, tandis que `n_blocks` agit sur l'axe horizontal (J/image).

En synthèse, fANOVA identifie `n_blocks` comme levier prioritaire sur l'axe efficacité (J/image), `batch_size` comme levier secondaire, et un impact négligeable des autres hyperpara-

mètres dans ce protocole. Côté exactitude, `lr_backbone` est le déterminant principal, tandis que `n_blocks` n'y joue qu'un rôle mineur (et `batch_size` reste marginal). Cela justifie la réduction de l'espace de recherche à `n_blocks`, `lr_backbone`, `batch_size`.

8 Conclusion

Ce mémoire visait à montrer comment l'optimisation multiobjectif des hyperparamètres, appliquée à un CNN en transfer learning, permet de construire un front de Pareto entre exactitude et efficacité énergétique (J/image) et d'en tirer un compromis adapté au cas d'étude. Deux questions ont guidé le travail : (i) comment construire et exploiter ce front pour la décision ? (ii) quels hyperparamètres déplacent le plus les solutions le long des axes du front ?

L'optimisation NSGA-II produit un front de Pareto concave à rendements décroissants : pour gagner en exactitude, un coût supplémentaire énergétique apparaît. Entre 0,06 et 0,10 J/image, 0,04 J supplémentaire apportent environ 2,5 points d'exactitude, alors qu'au-delà de 0,12 J/image, le même surcoût n'offre plus qu'environ 0,6 point. Inversement, réduire l'énergie par image entraîne une perte de performance. Le point de genou constitue un compromis particulièrement intéressant : il atteint 98 % de l'exactitude maximale pour 31 % d'énergie en moins par image. Ainsi, la première question de recherche trouve sa réponse : le front rend le compromis explicite et guide la sélection selon les contraintes comme par exemple un seuil d'exactitude minimal, un budget énergétique. Un entraînement haute fidélité valide ce compromis et la stabilité des tendances observées, sans en modifier les conclusions. La méthodologie propose un protocole reproductible et directement utilisable pour choisir une configuration.

L'analyse d'importance fANOVA indique, du côté du coût énergétique par image, que le nombre de blocs dégelés du modèle domine très nettement. Cet hyperparamètre, propre au TL, représente 93 % de la variance, car il fixe la profondeur de la backpropagation et donc directement le volume d'opérations nécessaires par image. La batch size quant à elle n'a qu'un effet plus modéré (7 %). Concernant l'exactitude, elle est principalement influencée par le learning rate appliqué au backbone préentraîné, qui explique 91 % de la variance, tandis que le nombre de blocs dégelés n'en explique que 6 %. En pratique, il est possible de restreindre l'espace de recherche des hyperparamètres aux seules variables les plus impactantes : learning rate du backbone, batch size et nombre de blocs dégelés. Grâce à cela, on peut orienter le front de Pareto selon l'objectif visé. Ajuster le degré de dégèlement agit surtout sur l'axe énergie, tandis que pour influer sur l'axe d'exactitude, il convient en premier lieu de modifier le learning rate du backbone.

Ce cas d'étude comporte néanmoins des limites. D'une part, la robustesse statistique n'a pas été évaluée par répétition avec plusieurs seeds, ce qui empêche de quantifier précisément la variabilité due aux initialisations et à l'ordre des batchs. Cette vérification améliorerait la solidité des conclusions. Enfin, la configuration de NSGA-II et le budget d'évaluations, fixés par les contraintes matérielles, n'ont pas été soumis à une analyse de sensibilité.

En résumé, l'optimisation multiobjectif offre une méthode pratique pour rendre visibles et maîtriser les compromis entre exactitude et efficacité énergétique à l'entraînement. Associée à l'analyse d'importance, elle permet de passer d'un réglage empirique des hyperparamètres à un processus clair et adapté aux contraintes du problème.

Références

- S. AFAQ et S. RAO : Significance Of Epochs On Training A Neural Network. *International Journal of Scientific & Technology Research*, juin 2020.
- W. AFZAL et R. TORKAR : On the application of genetic programming for software engineering predictive modeling : A systematic review. *Expert Systems with Applications*, 38(9):11984–11997, sept. 2011. ISSN 0957-4174.
- T. AKIBA, S. SANO, T. YANASE, T. OHTA et M. KOYAMA : Optuna : A Next-generation Hyperparameter Optimization Framework. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD '19, p. 2623–2631, New York, NY, USA, juil. 2019. Association for Computing Machinery. ISBN 978-1-4503-6201-6.
- L. F. W. ANTHONY, B. KANDING et R. SELVAN : Carbontracker : Tracking and Predicting the Carbon Footprint of Training Deep Learning Models, juil. 2020.
- A. ASPERTI, D. EVANGELISTA et M. MARZOLLA : Dissecting FLOPs Along Input Dimensions for GreenAI Cost Estimations. In G. NICOSIA, V. OJHA, E. LA MALFA, G. LA MALFA, G. JANSEN, P. M. PARDALOS, G. GIUFFRIDA et R. UMETON, éds : *Machine Learning, Optimization, and Data Science*, p. 86–100, Cham, 2022. Springer International Publishing. ISBN 978-3-030-95470-3.
- B. BAKER, O. GUPTA, N. NAIK et R. RASKAR : Designing Neural Network Architectures using Reinforcement Learning, mars 2017.
- J. BERGSTRA, B. KOMER, C. ELIASMITH, D. YAMINS et D. D. COX : Hyperopt : A Python library for model selection and hyperparameter optimization. *Computational Science & Discovery*, 8 (1):014008, juil. 2015. ISSN 1749-4699.
- J. CHEN, S.-h. KAO, H. HE, W. ZHUO, S. WEN, C.-H. LEE et S.-H. G. CHAN : Run, Don't Walk : Chasing Higher FLOPS for Faster Neural Networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, p. 12021–12031, 2023.
- X. CHEN : Escoin : Efficient Sparse Convolutional Neural Network Inference on GPUs, avr. 2019.
- Y. CHEN, W. LI, C. SAKARIDIS, D. DAI et L. VAN GOOL : Domain Adaptive Faster R-CNN for Object Detection in the Wild. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, p. 3339–3348, 2018.
- C. COLEMAN, D. NARAYANAN, D. KANG, T. ZHAO, J. ZHANG, L. NARDI, P. BAILIS, K. OLUKOTUN, C. RÉ et M. ZAHARIA : DAWN Bench : An End-to-End Deep Learning Benchmark and Competition.
- D. W. CORNE, N. R. JERRAM, J. D. KNOWLES et M. J. OATES : PESA-II : Region-based selection in evolutionary multiobjective optimization. In *Proceedings of the 3rd Annual Conference on Genetic and Evolutionary Computation*, GECCO'01, p. 283–290, San Francisco, CA, USA, juil. 2001. Morgan Kaufmann Publishers Inc. ISBN 978-1-55860-774-3.
- Z. C. DAGDIA et M. MIRCHEV : Chapter 15 - When Evolutionary Computing Meets Astro- and Geoinformatics. In P. ŠKODA et F. ADAM, éds : *Knowledge Discovery in Big Data from Astronomy and Earth Observation*, p. 283–306. Elsevier, jan. 2020. ISBN 978-0-12-819154-5.

- K. DEB, A. PRATAP, S. AGARWAL et T. MEYARIVAN : A fast and elitist multiobjective genetic algorithm : NSGA-II. *IEEE Transactions on Evolutionary Computation*, 6(2):182–197, avr. 2002. ISSN 1941-0026.
- R. DESISLAVOV, F. MARTÍNEZ-PLUMED et J. HERNÁNDEZ-ORALLO : Compute and Energy Consumption Trends in Deep Learning Inference. *Sustainable Computing : Informatics and Systems*, 38:100857, avr. 2023. ISSN 22105379.
- V. DUMOULIN et F. VISIN : A guide to convolution arithmetic for deep learning, jan. 2018.
- A. E. EIBEN et J. E. SMITH : *Introduction to Evolutionary Computing*. Natural Computing Series. Springer, Heidelberg, 2nd edition édn, 2015. ISBN 978-3-662-44874-8.
- Y. GAO, Y. RUAN, C. FANG et S. YIN : Deep learning and transfer learning models of energy consumption forecasting for a building with poor information data. *Energy and Buildings*, 223:110156, sept. 2020. ISSN 0378-7788.
- S. N. GOWDA, X. HAO, G. LI, S. N. GOWDA, X. JIN et L. SEVILLA-LARA : Watt For What : Rethinking Deep Learning’s Energy-Performance Relationship, sept. 2024.
- J. GUPTA, S. PATHAK et G. KUMAR : Deep Learning (CNN) and Transfer Learning : A Review. *Journal of Physics : Conference Series*, 2273(1):012029, mai 2022. ISSN 1742-6596.
- A. HIDAKA et T. KURITA : *Consecutive Dimensionality Reduction by Canonical Correlation Analysis for Visualization of Convolutional Neural Networks*, vol. 2017. déc. 2017.
- G. HINTON, O. VINYALS et J. DEAN : Distilling the Knowledge in a Neural Network, mars 2015.
- Y.-W. HONG, J.-S. LEU, M. FAISAL et S. W. PRAKOSA : Analysis of Model Compression Using Knowledge Distillation. *IEEE Access*, 10:85095–85105, 2022. ISSN 2169-3536.
- A. G. HOWARD, M. ZHU, B. CHEN, D. KALENICHENKO, W. WANG, T. WEYAND, M. ANDREETTO et H. ADAM : MobileNets : Efficient Convolutional Neural Networks for Mobile Vision Applications, avr. 2017.
- J. HOWARD et S. RUDER : Universal Language Model Fine-tuning for Text Classification, mai 2018.
- G. HU, Y. YANG, D. YI, J. KITTLER, W. CHRISTMAS, S. Z. LI et T. HOSPEDALES : When Face Recognition Meets With Deep Learning : An Evaluation of Convolutional Neural Networks for Face Recognition. *In Proceedings of the IEEE International Conference on Computer Vision Workshops*, p. 142–150, 2015.
- F. HUTTER, H. HOOS et K. LEYTON-BROWN : An Efficient Approach for Assessing Hyperparameter Importance. *In Proceedings of the 31st International Conference on Machine Learning*, p. 754–762. PMLR, jan. 2014a.
- F. HUTTER, H. HOOS et K. LEYTON-BROWN : An Efficient Approach for Assessing Hyperparameter Importance. *In Proceedings of the 31st International Conference on Machine Learning*, p. 754–762. PMLR, jan. 2014b.
- F. HUTTER, H. H. HOOS et K. LEYTON-BROWN : Sequential Model-Based Optimization for General Algorithm Configuration. *In C. A. C. COELLO, éd. : Learning and Intelligent Optimization*, p. 507–523, Berlin, Heidelberg, 2011. Springer. ISBN 978-3-642-25566-3.

M. IMAN, H. R. ARABNIA et K. RASHEED : A Review of Deep Transfer Learning and Recent Advancements. *Technologies*, 11(2):40, avr. 2023. ISSN 2227-7080.

S. JARKMAN, M. KARLBERG, M. POCEVIČIŪTĖ, A. BODÉN, P. BÁNDI, G. LITJENS, C. LUNDSTRÖM, D. TREANOR et J. VAN DER LAAK : Generalization of Deep Learning in Digital Pathology : Experience in Breast Cancer Metastasis Detection. *Cancers*, 14(21):5424, jan. 2022. ISSN 2072-6694.

I. KANDEL et M. CASTELLI : The effect of batch size on the generalizability of the convolutional neural networks on a histopathology dataset. *ICT Express*, 6(4):312–315, déc. 2020. ISSN 2405-9595.

A. KHOSLA, N. JAYADEVAPRAKASH, B. YAO et F.-F. LI : Novel Dataset for Fine-Grained Image Categorization : Stanford Dogs.

J. D. KNOWLES et D. W. CORNE : Approximating the Nondominated Front Using the Pareto Archived Evolution Strategy. *Evolutionary Computation*, 8(2):149–172, juin 2000. ISSN 1063-6560.

P. W. KOH, S. SAGAWA, H. MARKLUND, S. M. XIE, M. ZHANG, A. BALSUBRAMANI, W. HU, M. YASUNAGA, R. L. PHILLIPS, I. GAO, T. LEE, E. DAVID, I. STAVNESS, W. GUO, B. EARNSHAW, I. HAQUE, S. M. BEERY, J. LESKOVEC, A. KUNDAJE, E. PIERSON, S. LEVINE, C. FINN et P. LIANG : WILDS : A Benchmark of in-the-Wild Distribution Shifts. In *Proceedings of the 38th International Conference on Machine Learning*, p. 5637–5664. PMLR, juil. 2021.

D. LI, X. CHEN, M. BECCHI et Z. ZONG : Evaluating the Energy Efficiency of Deep Convolutional Neural Networks on CPUs and GPUs. p. 477–484, oct. 2016.

X. LI, M. CEN, J. XU, H. ZHANG et X. S. XU : Improving Feature Extraction from Histopathological Images Through A Fine-tuning ImageNet Model, jan. 2022.

Z. LI, F. LIU, W. YANG, S. PENG et J. ZHOU : A Survey of Convolutional Neural Networks : Analysis, Applications, and Prospects. *IEEE Transactions on Neural Networks and Learning Systems*, PP:1–21, juin 2021.

D. LIAN, D. ZHOU, J. FENG et X. WANG : Scaling & Shifting Your Features : A New Baseline for Efficient Model Tuning. *Advances in Neural Information Processing Systems*, 35:109–123, déc. 2022.

T. LIANG, J. GLOSSNER, L. WANG, S. SHI et X. ZHANG : Pruning and quantization for deep neural network acceleration : A survey. *Neurocomputing*, 461:370–403, oct. 2021. ISSN 0925-2312.

Y. LIU, Y. KANG, T. ZOU, Y. PU, Y. HE, X. YE, Y. OUYANG, Y.-Q. ZHANG et Q. YANG : Vertical Federated Learning : Concepts, Advances and Challenges. *IEEE Transactions on Knowledge and Data Engineering*, 36(7):3615–3634, juil. 2024. ISSN 1041-4347, 1558-2191, 2326-3865.

I. LOSHCHILOV et F. HUTTER : Decoupled Weight Decay Regularization, jan. 2019.

V. P. C. MAGBOO et M. S. A. MAGBOO : Batch Size Selection in Convolutional Neural Networks for Glaucoma Classification. *Procedia Computer Science*, 235:2749–2755, jan. 2024. ISSN 1877-0509.

J. PLESTED et T. GEDEON : Deep transfer learning for image classification : A survey, mai 2022.

R. PRAMODITHA : Classification of Neural Network Hyperparameters, sept. 2022.

L. PRECHELT : Early Stopping - But When ? In G. B. ORR et K.-R. MÜLLER, éds : *Neural Networks : Tricks of the Trade*, p. 55–69. Springer, Berlin, Heidelberg, 1998. ISBN 978-3-540-49430-0.

I. RATKOVIĆ, N. BEŽANIĆ, O. S. ÜNSAL, A. CRISTAL et V. MILUTINOVIC : Chapter One - An Overview of Architecture-Level Power- and Energy-Efficient Design Techniques. In A. R. HURSON, éd. : *Advances in Computers*, vol. 98, p. 1–57. Elsevier, jan. 2015.

S. ROY : Understanding the Impact of Post-Training Quantization on Large Language Models. <https://arxiv.org/abs/2309.05210v3>, sept. 2023.

V. SATOPAA, J. ALBRECHT, D. IRWIN et B. RAGHAVAN : Finding a "Kneedle" in a Haystack : Detecting Knee Points in System Behavior. In *2011 31st International Conference on Distributed Computing Systems Workshops*, p. 166–171, juin 2011.

F. M. SHIRI, T. PERUMAL, N. MUSTAPHA et R. MOHAMED : A Comprehensive Overview and Comparative Analysis on Deep Learning Models : CNN, RNN, LSTM, GRU, oct. 2024.

B. SHIROKIKH, I. ZAKAZOV, A. CHERNYAVSKIY, I. FEDULOVA et M. BELYAEV : First U-Net Layers Contain More Domain Specific Information Than The Last Ones, août 2020.

Z. N. K. SWATI, Q. ZHAO, M. KABIR, F. ALI, Z. ALI, S. AHMED et J. LU : Brain tumor classification for MR images using transfer learning and fine-tuning. *Computerized Medical Imaging and Graphics*, 75:34–46, juil. 2019. ISSN 0895-6111.

M. TAN et Q. LE : EfficientNet : Rethinking Model Scaling for Convolutional Neural Networks. In *Proceedings of the 36th International Conference on Machine Learning*, p. 6105–6114. PMLR, mai 2019.

M. TAN et Q. LE : EfficientNetV2 : Smaller Models and Faster Training. In *Proceedings of the 38th International Conference on Machine Learning*, p. 10096–10106. PMLR, juil. 2021a.

M. TAN et Q. V. LE : EfficientNetV2 : Smaller Models and Faster Training, juin 2021b.

D. THEODORAKOPOULOS, F. STAHL et M. LINDAUER : Hyperparameter Importance Analysis for Multi-Objective AutoML. oct. 2024.

A. TSCHAND, A. T. R. RAJAN, S. IDGUNJI, A. GHOSH, J. HOLLEMAN, C. KIRALY, P. AMBAL-KAR, R. BORKAR, R. CHUKKA, T. COCKRELL, O. CURTIS, G. FURSIN, M. HODAK, H. KASSA, A. LOKHMOTOV, D. MISKOVIC, Y. PAN, M. P. MANMATHAN, L. RAYMOND, T. S. JOHN, A. SURESH, R. TAUBITZ, S. ZHAN, S. WASSON, D. KANTER et V. J. REDDI : MLPerf Power : Benchmarking the Energy Efficiency of Machine Learning Systems from μ Watts to MWatts for Sustainable AI. In *2025 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, p. 1201–1216, mars 2025.

I. A. USMANI, M. T. QADRI, R. ZIA, F. S. ALRAYES, O. SAIDANI et K. DASHTIPOUR : Interactive Effect of Learning Rate and Batch Size to Implement Transfer Learning for Brain Tumor Classification. *Electronics*, 12(4):964, jan. 2023. ISSN 2079-9292.

- N. VIJAYKUMAR, G. PEKHIMENKO, A. JOG, S. GHOSE, A. BHOWMICK, R. AUSAVARUNGNIRUN, C. DAS, M. KANDEMIR, T. C. MOWRY et O. MUTLU : Chapter 15 - A framework for accelerating bottlenecks in GPU execution with assist warps. In H. SARBAZI-AZAD, éd. : *Advances in GPU Research and Practice*, Emerging Trends in Computer Science and Applied Computing, p. 373–415. Morgan Kaufmann, Boston, jan. 2017. ISBN 978-0-12-803738-6.
- T. WANG, D. J. WU, A. COATES et A. Y. NG : End-to-end text recognition with convolutional neural networks. In *Proceedings of the 21st International Conference on Pattern Recognition (ICPR2012)*, p. 3304–3308, nov. 2012.
- Z. WANG et G. P. RANGAIAH : Application and Analysis of Methods for Selecting an Optimal Solution from the Pareto-Optimal Front obtained by Multiobjective Optimization. *Industrial & Engineering Chemistry Research*, 56(2):560–574, jan. 2017. ISSN 0888-5885.
- Z. WANG, Y. PEI et J. LI : A Survey on Search Strategy of Evolutionary Multi-Objective Optimization Algorithms. *Applied Sciences*, 13(7):4643, jan. 2023. ISSN 2076-3417.
- WIKIPEDIA CONTRIBUTORS : Max pooling, juin 2025.
- M. WOJCIUK, Z. SWIDERSKA-CHADAJ, K. SIWEK et A. GERTYCH : Improving classification accuracy of fine-tuned CNN models : Impact of hyperparameter optimization. *Heliyon*, 10(5), mars 2024. ISSN 2405-8440.
- Y. XU, S. MARTÍNEZ-FERNÁNDEZ, M. MARTINEZ et X. FRANCH : Energy Efficiency of Training Neural Network Architectures : An Empirical Study. <https://arxiv.org/abs/2302.00967v1>, fév. 2023.
- W. YAN, Y. WANG, S. GU, L. HUANG, F. YAN, L. XIA et Q. TAO : The Domain Shift Problem of Medical Image Segmentation and Vendor-Adaptation by Unet-GAN. In D. SHEN, T. LIU, T. M. PETERS, L. H. STAIB, C. ESSERT, S. ZHOU, P.-T. YAP et A. KHAN, éds : *Medical Image Computing and Computer Assisted Intervention – MICCAI 2019*, p. 623–631, Cham, 2019. Springer International Publishing. ISBN 978-3-030-32245-8.
- G. YANG, S. YU, Y. SHENG et H. YANG : Attention and feature transfer based knowledge distillation. *Scientific Reports*, 13(1):18369, oct. 2023. ISSN 2045-2322.
- Z. YANG et W. ARMOUR : The Hidden Joules : Evaluating the Energy Consumption of Vision Backbones for Progress Towards More Efficient Model Inference. In *Forty-Second International Conference on Machine Learning*, juin 2025.
- H. YAO, Z. XU, Y. HOU, Q. DONG, P. LIU, Z. YE, X. PEI, M. OESER, L. WANG et D. WANG : Advanced industrial informatics towards smart, safe and sustainable roads : A state of the art. *Journal of Traffic and Transportation Engineering (English Edition)*, 10(2):143–158, avr. 2023. ISSN 2095-7564.
- J. YOU, J.-W. CHUNG et M. CHOWDHURY : Zeus : Understanding and Optimizing GPU Energy Consumption of DNN Training.
- Q. ZHANG et H. LI : MOEA/D : A Multiobjective Evolutionary Algorithm Based on Decomposition. *IEEE Transactions on Evolutionary Computation*, 11(6):712–731, déc. 2007. ISSN 1941-0026.
- F. ZHUANG, Z. QI, K. DUAN, D. XI, Y. ZHU, H. ZHU, H. XIONG et Q. HE : A Comprehensive Survey on Transfer Learning, juin 2020.

E. ZITZLER et S. KÜNZLI : Indicator-Based Selection in Multiobjective Search. In X. YAO, E. K. BURKE, J. A. LOZANO, J. SMITH, J. J. MERELLO-GUERVÓS, J. A. BULLINARIA, J. E. ROWE, P. TIÑO, A. KABÁN et H.-P. SCHWEFEL, éds : *Parallel Problem Solving from Nature - PPSN VIII*, p. 832–842, Berlin, Heidelberg, 2004. Springer. ISBN 978-3-540-30217-9.

E. ZITZLER, M. LAUMANNS et L. THIELE : SPEA2 : Improving the strength pareto evolutionary algorithm. Report, ETH Zurich, Computer Engineering and Networks Laboratory, mai 2001.