Note méthodologique : preuve de concept

Dataset retenu

Pour cette preuve de concept, je me suis appuyé sur le Stanford Dogs Dataset, qui contient au total 20 580 images réparties en 120 races de chiens. Dû à un manque de puissance de calcul sur ma machine, j'ai réduit le scope du projet en sélectionnant uniquement les 10 classes les plus représentées (entre 210 et 250 images chacune). Ces 10 races sont très diverses, certaines sont très différentes (Shi-Tzu & Leonberg par exemple) tandis que d'autres sont beaucoup plus proches (Irish & Scottish Hound par exemple). Les chiens sont photographiés sous des angles variés, dans différentes postures et sur des arrière-plans très hétérogènes.

Pour évaluer rigoureusement mon modèle, j'ai opéré un découpage stratifié du sous-ensemble retenu selon la répartition suivante :

- 80 % des images pour l'entraînement (1785 images)
- 10 % pour la validation (224 images)
- 10 % pour le test (224 images)

La stratification garantit que chaque classe est représentée de façon proportionnelle dans chacun des splits, et l'utilisation d'une graine fixe assure la reproductibilité. Cette configuration nous permet de mesurer à la fois la capacité d'apprentissage, le réglage des hyperparamètres et la généralisation sur un ensemble équilibré et homogène.

Les concepts de l’algorithme récent

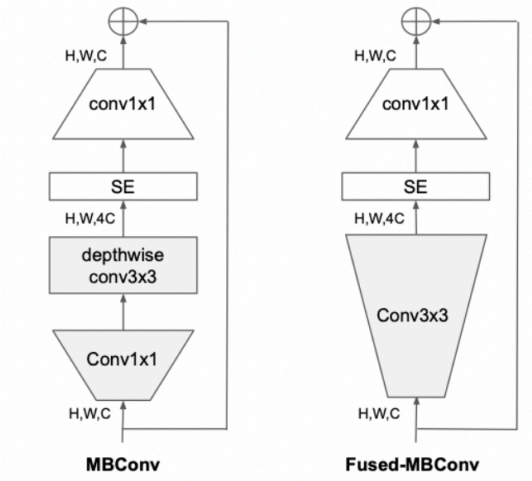
À l’origine, j’envisageais d’exploiter des Vision Transformers (ViT) pour leur capacité à modéliser les interactions à longue portée au sein de l’image grâce à leurs mécanismes d’attention. Cependant, les ViT présentent deux inconvénients majeurs dans notre contexte :

1. **Besoin en données important :** les Vision Transformers, nécessitent généralement des centaines de milliers d’images pour converger vers une bonne généralisation. Avec seulement ~230 images par classe, le risque d’overfitting et de résultats irréguliers était trop élevé.
2. **Coût de calcul élevé :** Le mécanisme d’attention allonge considérablement les temps d’entraînement et fait exploser la consommation mémoire.

Pour pallier à ces limitations, je me suis tourné vers **EfficientNetV2**, qui est bien plus performant sur un dataset de cette taille. Ce modèle exploite 2 nouveaux concepts :

1. Les blocs Fused-MBConv

Dans EfficientNetV1, le bloc de base MBConv se compose de trois étapes distinctes : expansion par convolution pointwise, convolution depthwise, puis projection pointwise. EfficientNetV2 introduit le Fused-MBConv, qui combine les deux premières opérations (expansion + depthwise) en une unique convolution 3×3. En réduisant le nombre d’opérations séparées et les changements de format de tenseurs, on limite les appels GPU, ce qui accélère sensiblement l’entraînement.



2. Training-aware Neural Architecture Search (NAS)

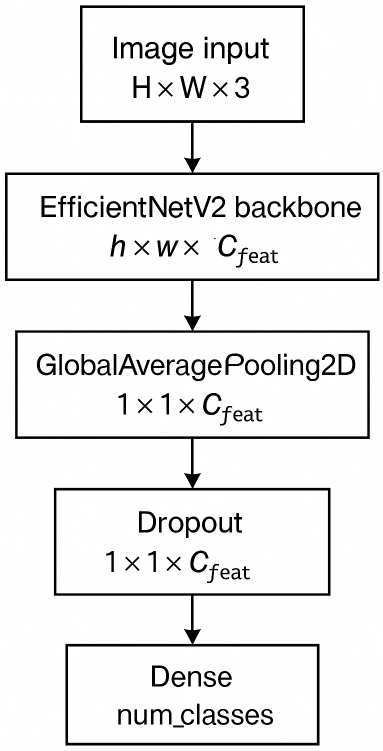
Plutôt que d'optimiser uniquement le nombre de FLOPs, le NAS « training-aware » intègre directement le temps d'entraînement comme critère d'optimisation.

Il ajuste automatiquement la profondeur (nombre de couches), la largeur (nombre de canaux par couche) et la résolution d'entrée du modèle pour atteindre un compromis optimal entre précision et vitesse de convergence. Grâce à son architecture, le modèle reste suffisamment léger tout en gardant assez de capacité pour distinguer les différences subtiles entre les races, sans avoir besoin d'un long pré-entraînement.

En synthèse, EfficientNetV2 nous permet de conserver une architecture moderne et performante, tout en respectant les contraintes de reproductibilité, de temps de calcul et de quantité de données disponibles, assurant ainsi un entraînement rapide et stable sur notre jeu de 2 300 images.

La modélisation

Dans mon approche, j'ai choisi la version EfficientNetV2-S pré-entraîné sur ImageNet, dont j'ai dégelé l'intégralité des couches pour procéder à un fine-tuning complet. L'image d'entrée, redimensionnée à 224×224 pixels et normalisée selon la moyenne et l'écart-type calculés sur le dataset, traverse d'abord le backbone EfficientNetV2-S suivi ensuite d'une couche de GlobalAveragePooling2D. Pour prévenir l'over-fitting, le vecteur de sortie passe ensuite par une couche de dropout, dont le taux est déterminé par une recherche par grille d'hyperparamètres. Enfin, une couche Dense à dix sorties (autant que de classes) suit une activation



softmax pour produire les probabilités de prédiction.

L'entraînement s'effectue avec l'optimiseur Adam, dont le learning rate initial est ajusté par grid search, et soumis à un scheduler à décroissance cosinusoïdale, permettant une réduction progressive et lissée du taux d'apprentissage au fil des itérations. Le batch size, le nombre d'époques et le taux de dropout sont également optimisés par grid search. La configuration optimale est celle qui maximise simultanément l'accuracy et le F1-score sur l'ensemble de validation tout en minimisant la validation loss.

La richesse et la robustesse du modèle reposent également sur une pipeline d'augmentations, implémentée avec la bibliothèque Albumentations. Chaque image est redimensionnée, enrichie par un égaliseur adaptatif CLAHE, normalisée, puis soumise à des transformations géométriques (translations, zooms ± 20 %, rotations ± 40 °, flips), à des ajustements d'éclairage (luminosité et contraste ± 20 %), et à un léger flou médian, avant d'être convertie en tenseur PyTorch. Chacune de ces augmentations a une probabilité plus ou moins élevée de s'appliquer sur chaque image.

À titre de référence, j'ai utilisé en baseline le modèle de production de mon projet précédent : un ResNet-50 pré-entraîné sur ImageNet, dont toutes les couches ont été dégelées pour un fine-tuning complet sur notre jeu de dix races. Pour garantir une comparaison équitable, ce ResNet-50 partage avec notre EfficientNetV2-S l'ensemble de la pipeline d'augmentations. Les hyperparamètres de ce modèle sont issus d'une précédente grid search. Notre objectif avec cette preuve de concept est de démontrer que, malgré sa compacité accrue et son coût de calcul réduit, EfficientNetV2-S est capable de dépasser la performance du ResNet-50.

Pour évaluer les modèles, j'analyse leur performance sur l'ensemble de test à travers plusieurs indicateurs : la précision Top-1 et Top-3, le F1-score macro, une matrice de confusion détaillée pour identifier les confusions récurrentes entre races visuellement proches, ainsi que le temps moyen d'inférence par image et le nombre total de paramètres du réseau. En plus de ces mesure, je visualise également les courbes

d'évolution de la loss et de l'accuracy, afin d'illustrer la convergence du modèle et de détecter d'éventuels déséquilibres d'apprentissage.

Une synthèse des résultats

Grâce à notre étude comparative, il apparaît que EfficientNetV2-S se distingue dans la plupart des mesures. Sa convergence rapide lors de l'entrainement, en seulement 12 min 32 s contre 15 min 05 s pour ResNet-50 et par une meilleure qualité de généralisation : sa perte de validation est nettement inférieure à celle de ResNet-50, tandis que ses mesures de précision culminent à plus de 95 % en Top-1 (contre 89,29 % pour ResNet) et à 99,55 % en Top-3 (contre 97,32 % pour ResNet). Son F1-score macro atteint 0,9321, traduisant une excellente cohérence sur l'ensemble des dix races, là où ResNet-50 plafonne à 0,8708. Sur GPU, cette supériorité se confirme par un temps d'inférence moyen de 168 ms contre 219 ms enregistrées pour ResNet-50. Cependant, ResNet-50 conserve un très net avantage lorsqu'on bascule l'inférence sur CPU.

Tableau comparatif :

Métrique	ResNet-50	EfficientNetV2S
Validation loss	0,405	**0,184**
Validation Top-1 Accuracy (%)	89,29 %	**95,54 %**
Validation Top-3 Accuracy (%)	97,32 %	**99,55 %**
F1-score macro	0,8708	**0,9321**
Inference time (GPU, ms)	219 ms	**168 ms**
Temps d'entraînement	15 min 05 s	**12 min 32 s**

Au cours de cette expérimentation, EfficientNetV2-S s'est imposé comme le modèle le plus performant sur les critères de précision et d'efficacité GPU : il converge plus rapidement à l'entraînement, affiche une perte de validation nettement plus faible, et obtient les meilleurs scores de Top-1, Top-3 et F1-macro. En revanche, lorsque l'inférence est effectuée sur CPU, ResNet-50 conserve un net avantage en latence,

offrant un temps moyen de traitement bien plus rapide que celui d'EfficientNetV2-S. Ce constat suggère de privilégier EfficientNetV2-S pour les environnements dotés de GPU et d'un fort besoin de précision, et d'opter pour ResNet-50 pour des environnement dotés uniquement de CPU.

Les limites et les améliorations possibles

Bien que EfficientNetV2-S excelle par sa rapidité d'entraînement et par la qualité de ses prédictions, son architecture sophistiquée repose sur des blocs fusionnés et des opérations de depth-wise convolution hautement optimisées pour les processeurs graphiques. En l'absence de GPU, ces mêmes blocs se traduisent par une latence d'inférence sur CPU très élevée, rendant le modèle peu adapté aux contextes embarqués ou aux environnements à ressources limitées. Cette dépendance matérielle limite considérablement la flexibilité du déploiement : on ne peut plus garantir une expérience utilisateur fluide sur un simple CPU, ce qui est problématique pour des applications mobiles par exemple, où l'accès à un GPU n'est pas systématique. De plus, le jeu de données utilisé pour l'entraînement ne comprend qu'environ 2300 images réparties sur dix classes, ce qui reste relativement restreint pour couvrir l'ensemble de la variabilité rencontrée en situation réelle. Même si la perte de validation reste faible et les scores Top-1 et Top-3 sont élevés, il est possible que le modèle ait mémorisé certaines caractéristiques spécifiques du jeu de donnée, au détriment de sa capacité à généraliser sur de nouvelles images. Pour pallier ce risque, il est indispensable d'enrichir le jeu de données et de mettre en place des techniques d'augmentation avancée.
Parmi les pistes d'amélioration envisageables, le progressive learning se distingue également. Il consiste en démarrant l'entraînement sur des images redimensionnées à faible résolution, avec un minimum d'augmentations, afin que le réseau saisisse d'abord les structures globales. La résolution des entrées ainsi que l'intensité des transformations appliquées sont ensuite augmentées graduellement.

Cette montée progressive en difficulté, décrite dans l'article **« EfficientNetV2: Smaller Models and Faster Training »** (de Mingxing Tan et Quoc V. Le), accélère la convergence, diminue la durée totale d'apprentissage et accroît la performance du modèle.