PROOF OF CONCEPT

ÉTUDE DE LA MÉTHODE "LEARNING WHAT AND WHERE TO TRANSFER"

Sylvain Friot

Novembre 2020

Résumé

La méthode "Learning What and Where To Transfer" a été publié par Yunhun Jang, Hankook Lee et Sung Ju Hwang, Jinwoo Shin en mai 2019. Elle a été présentée à l'ICML (International Conference on Machine Learning) à Long-Beach (Californie, Etats-Unis) en juin 2019.
La lecture du papier laisse entrevoir une méthode aux performances intéressantes pour la classification d'images lorsque le nombre de données est limité. Selon les auteurs, elle doit pouvoir être appliquée à tous types de réseau. Elle consiste à transférer les connaissances d'un réseau complexe à un réseau plus simple, à travers une méthode inspirée de la knowledge distillation et d'autres études autour de ce principe. Elle détermine grâce à des meta-networks quelle information doit être transmise d'une couche donnée du réseau source vers une couche donnée du réseau cible, et dans quelle quantité.

Je décide d'étudier son apport au projet de classification d'images sur le Stanford Dogs Dataset que j'ai traité précédemment.
J'utilise donc en priorité les réseaux qui étaient ressortis de ce projet : un réseau convolutif très simple peu performant et le réseau EfficientNetB4. J'utilise également VGG16 qui est le réseau le plus simple étudié lors de ce précédent projet. Pour finir, j'étends mon étude à deux autres développements d'EfficientNet : les versions B0 et B7.
J'étudie si cette méthode est capable de faire progresser un réseau très simple à partir du réseau EfficientNetB4 très efficace en terme d'accuracy. J'étudie également le transfert de connaissances entre deux réseaux aux architectures très différentes : d'EfficientNetB4 vers VGG16. Enfin, j'étudie le transfert de connaissances d'EfficientNetB7 vers EfficientNetB0.

Mes résultats montrent que la méthode est particulièrement gourmande en ressources et ne se généralise pas à tous les types de réseaux.

1. Thématique

Ce projet reprend la problématique de la détermination de la race d'un chien à partir de sa photo, basée sur le jeu de données Stanford Dogs Dataset[1]. Je cherche à obtenir de meilleurs résultats avec des modèles plus simples.
La simplification du modèle final est importante lorsque la mise en production du modèle nécessite des estimations rapides pour de nouvelles données. En effet, plus le modèle est complexe, plus la prédiction de la classe est longue à obtenir.
Une accuracy élevée est difficile à obtenir avec un réseau de neurones convolutif simple dans un problème de classification d'images. Nous cherchons une méthode afin d'obtenir de meilleures performances avec un modèle simple.

2. Etat de l'art

2.1. Transfer Learning

Le Transfer Learning[2] est couramment utilisé pour améliorer la performance d'un problème de classification d'images, ainsi que pour d'autres tâches comme le Natural Language Processing.
Le Transfer Learning consiste à utiliser un réseau complexe pré-entraîné sur une grande quantité de données, si possible similaires aux nôtres, sur un problème général, si possible proche de notre tâche. Ce réseau pré-entraîné est le modèle source.
Pour obtenir le modèle cible adapté à notre problématique, il suffit de remplacer la couche de classification par une couche adaptée à notre tâche de classification, et d'entraîner seulement cette dernière.

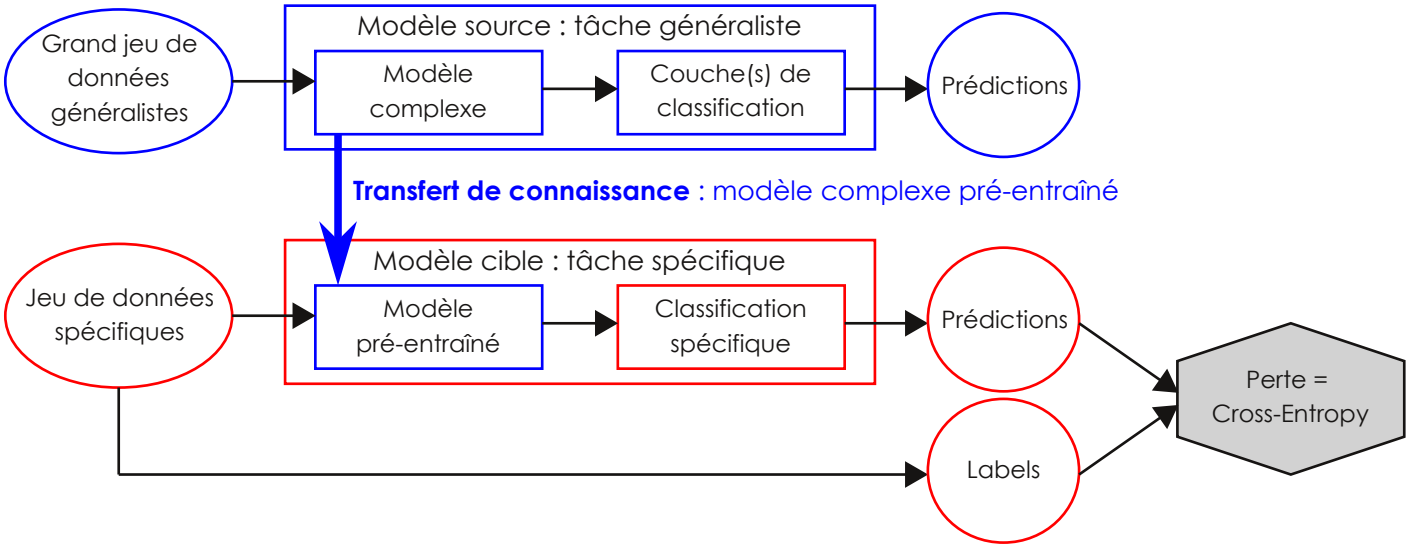


Figure 1 : Le principe du Transfer Learning

La performance peut encore être améliorée grâce au fine-tuning : ré-entraîner une ou plusieurs couches du modèle pré-entraîné avec un taux d'apprentissage faible. Le nombre de couches ré-entraînées dépend :
• du nombre de données disponibles pour l'apprentissage de notre tâche ;
• et de la similarité de nos données avec celles du modèle pré-entraîné.

Deux reproches sont adressés au Transfer Learning. D'une part, le gain de performance dépend de la similarité entre la tâche généraliste et la tâche spécifique. D'autre part, la performance du modèle final tend à être corrélée à la complexité du modèle transféré (profondeur et largeur du réseau). Le Transfer Learning tend donc à privilégier des modèles complexes. Ces modèles ont un long temps de calcul lors de l'estimation de la classe d'une nouvelle image, ce qui est problématique en production. D'autres approches ont été développées pour remédier à ces limitations.

2.2. Knowledge Distillation

Le principe de la Knowledge Distillation a été introduit en 2015 par Hinton, Vinyals et Dean[3]. Il vise à obtenir un modèle final plus simple.
Il met en relation un modèle source complexe (le professeur) et un modèle cible plus compact (l'élève). Ces deux modèles sont entraînés sur la même tâche et les mêmes données. L'objectif est d'apprendre au modèle cible à produire des estimations les plus proches possibles de celles du modèle source.
Pour cela, les prédictions des deux modèles sont calculées à l'aide d'une fonction softmax avec une température élevée.

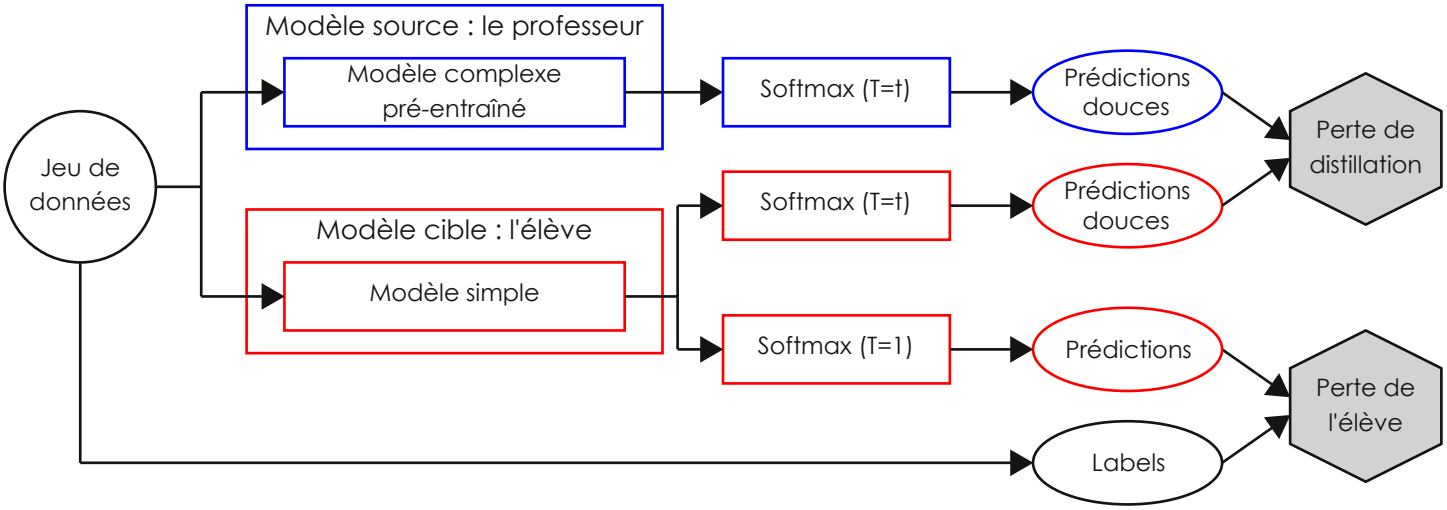
1. Stanford Dogs Dataset : 20580 images de chiens de 120 races différentes, issues de la base de données ImageNet, disponibles sur le site de Stanford http://vision.stanford.edu/aditya86/ImageNetDogs/main.html
2. https://towardsdatascience.com/transfer-learning-from-pre-trained-models-f2393f124751
3. Geoffrey Hinton, Oriol Vinyals, Jeff Dean : Distilling the Knowledge in a Neural Network, *Deep Learning and Representation Learning Workshop, Advances in Neural Information Processing Systems 29 (NIPS 2015)*, 2015.

Plus la température d'une fonction softmax est élevée, plus les probabilités des différentes classes sont lissées et proches les unes des autres. La température est un hyperparamètre de cette approche. La première partie de la perte de cette approche est la Cross Entropy calculée avec ces prédictions "douces" des deux modèles.

Softmax avec température : $P^{\tau}(a) = \frac{e^{(\frac{\hat{y}(a)}{\tau})}}{\sum_{i=1}^{n} e^{(\frac{\hat{y}(i)}{\tau})}}$ avec : n : nombre de classes possibles
τ : température
$\hat{y}$: sortie du neurone i de la couche dense de classification

Equation 1 : La fonction softmax avec une température

Les prédictions du modèle cible sont également calculées sur la base d'une fonction softmax classique, avec une température de 1. La deuxième partie de la perte est la Cross Entropy calculée entre ces prédictions et les labels du jeu de données.
La Knowledge Distillation cherche donc à orienter les poids du modèle cible de façon à ce qu'il prédise de la manière la plus proche possible du modèle source.



Perte totale = Cross-Entropy(labels, P^{1}_{cible}) + λ x Cross-Entropy(P^{τ}_{source} , P^{τ}_{cible}) où λ est un hyperparamètre qui permet de moduler l'importance des deux pertes

Figure 2 : Le principe de la Knowledge Distillation

2.3. Learning without Forgetting

La Knowledge Distillation présente une contrainte forte : les deux modèles doivent traiter la même tâche. L'approche Learning without Forgetting a été proposée en 2018 par Li et Hoiem[4] afin de permettre une application sur une tâche différente.

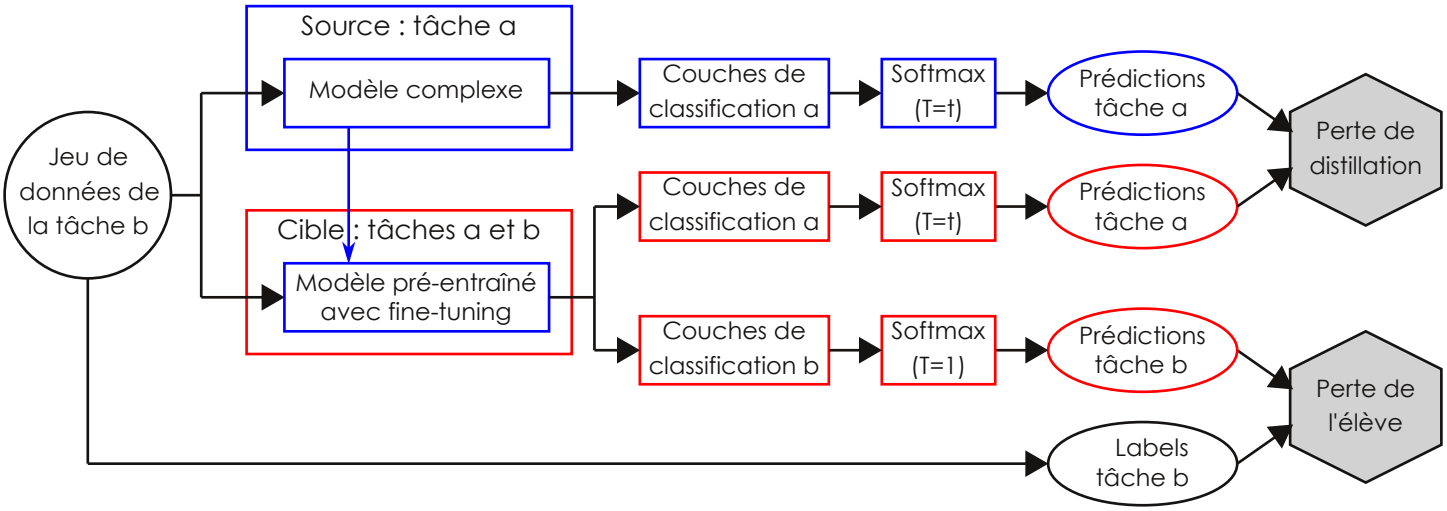


Figure 3 : Le principe du Learning without Forgetting - apprendre une nouvelle tâche sans oublier l'ancienne tâche.

La partie d'extraction des features est séparée de la partie spécifique à l'apprentissage de la tâche (classification). Le modèle élève est basé sur le modèle source auquel on ajoute une partie de traitement spécifique à la nouvelle tâche. On

4. Zhizhong Li, Derek Hoiem : Learning wihout Forgetting, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2018.

applique un fine-tuning de toutes les couches issues du modèle source pré-entraîné et un entraînement de la partie spécifique à la nouvelle tâche.
La perte totale est composée d'une perte de distillation et d'une perte spécifique à la nouvelle tâche. La perte de distillation est calculée entre les prédictions du modèle source et celles du modèle cible sur la tâche initiale. La perte de l'élève est estimée sur les prédictions de la nouvelle tâche.

2.4. FitNets : Feature Matching

Le Feature Matching a été présenté par Romero et co en 2015[5]. Il vise à améliorer la Knowledge Distillation en guidant l'élève dans son processus d'apprentissage.
Ils introduisent l'idée d'une perte intermédiaire qui compare la sortie d'une couche intermédiaire du modèle source et du modèle cible. La minimisation de cette perte tend donc à ce que la couche intermédiaire du modèle cible apprenne à extraire des feature maps similaires à la couche intermédiaire du modèle source.
Cet apprentissage d'une couche intermédiaire, en sus de l'apprentissage de la seule prédiction finale, représente une sorte de régularisation. Ils choisissent d'effectuer cette apprentissage sur la couche intermédiaire du milieu des deux réseaux, pour que la régularisation soit suffisamment forte tout en laissant suffisamment de latitude d'apprentissage au réseau cible.

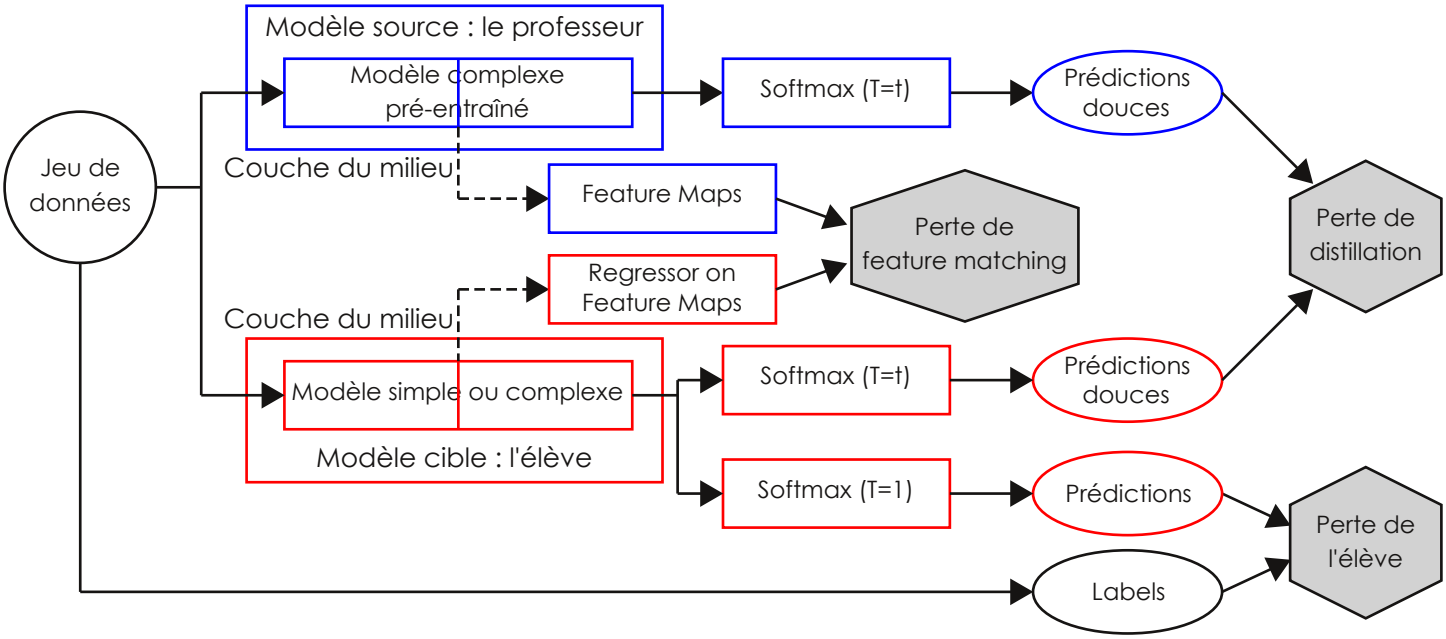


Figure 4 : Le principe du Feature Matching des FitNets

La régression appliquée sur les feature maps du réseau cible sert ramener la taille (largeur x hauteur) des feature maps du modèle cible à celle des feature maps du modèle source. Afin de limiter le nombre de paramètre, on utilise une convolution comme régresseur.
La perte de feature matching est la perte l2 entre les sorties des feature maps de la couche intermédiaire du modèle source et la transformation linéaire (régression par convolution) des sorties des features maps de la couche intermédiaire du modèle cible = (1/2) * ||Sorties des feature maps de la source - convolution(Sorties des feature maps de la Cible)||²
L'entraînement se déroule en deux étapes : un pré-entraînement de la couche guidée du modèle cible, basé sur la minimisation de la perte de feature matching, puis un entraînement du réseau cible entier basé sur la perte de distillation et la perte de l'élève (entraînement classique de Knowledge Distillation).

2.5. Attention Transfer

L'Attention Transfer reprend l'idée de la Knowledge Distillation en basant le calcul de la perte de distillation sur des attention maps intermédiaires. Il a été proposé par Zagoruyko et Komodakis[6] en 2017. L'objectif est de guider l'apprentissage du modèle cible grâce aux couches intermédiaires, sans introduire de nouveaux paramètres comme le fait l'approche des FitNets.
Le principe est de calculer l'attention map en sortie d'une ou plusieurs couches du modèle professeur (source) et du modèle élève (cible). En minimisant la différence entre ces attention maps, on force le modèle cible à avoir des extractions de features similaires au modèle source dans une ou plusieurs couches intermédiaires.

Une attention map pour une couche donnée est obtenue en transformant la matrice 3D (hauteur x largeur x nombre de

5. Adriana Romero, Nicolas Ballas, Samira Ebrahimi Kahou, Antoine Chassang, Carlo Gatta, Yoshua Bengio : Fitnets, Hints for Thin Deep Nets, *The 3rd International Conference on Learning Representations* (ICLR 2015), 2015.
6. Sergey Zaogoruyko, Nikos Komodakis : Paying More Attention To Attention: Improving the Performance of Convolutional Neural Networks via Attention Transfer, *The 5th International Conference on Learning Representations* (ICLR 2017), 2017.

feature maps) de la sortie de cette couche en une matrice 2D de dimension hauteur x largeur.
Différentes fonctions peuvent être utilisées pour aggréger les activations des différentes feature maps : la somme des valeurs absolues, la somme de la puissance p des valeurs absolues (pour augmenter le poids des zones les plus discriminantes) ou le maximum des valeurs absolues à la puissance p (pour se focaliser sur l'activation la plus discriminante parmi les différentes feature maps).

$$F : R^{C\times H\times L} \rightarrow R^{H\times L}$$

$$F_{sum}(A) = \sum_{i=1}^{C} |A_i|$$

$$F^{p}_{sum}(A) = \sum_{i=1}^{C} |A_i|^p$$

$$F^{p}_{max}(A) = \max_{i=1,C} |A_i|^p$$

avec :
C = nombre de feature maps (channels) de la convolution
A = Activations (sorties) de la couche convolutionnelle
H x L = taille des feature maps (hauteur x largeur)

Equation 2 : Fonctions de calcul des attention maps

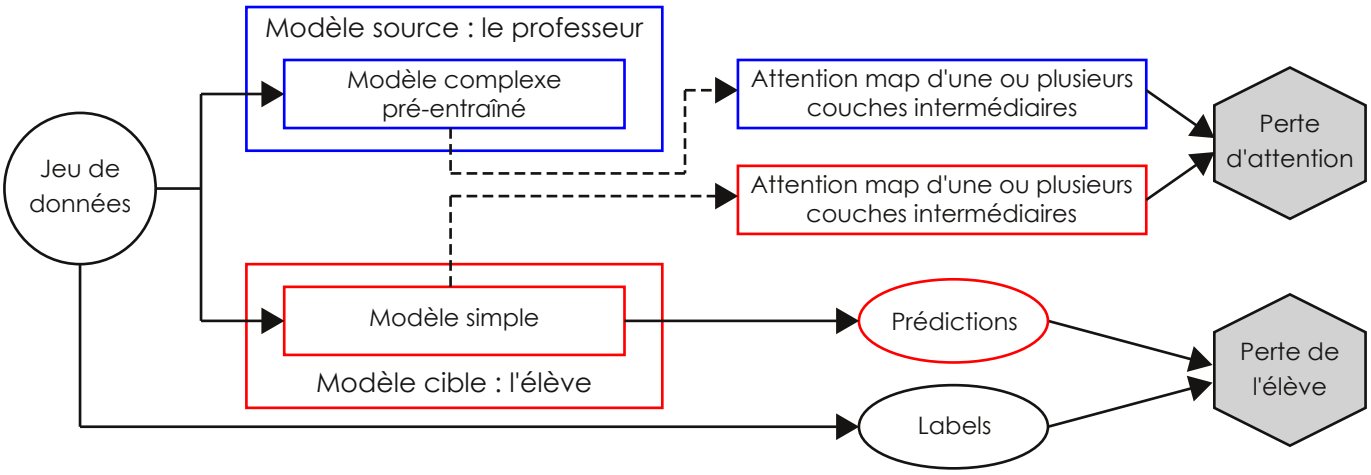


Figure 5 : Le principe de l'Attention Transfer

La perte d'attention est basée sur la perte l2 normalisée. La vectorisation de l'attention map est normalisée en la divisant par sa propre norme 2. Soit, si on note Q la vectorisation d'une Attention Map, et paires l'ensemble des paires de couches intermédiaires sélectionnées :

Perte totale = Cross-Entropy(labels, P^{1}_{cible}) + λ x $\frac{1}{2}\sum_{j\in paires} || \frac{Q^{j}_{cible}}{||Q^{j}_{cible}||_2} - \frac{Q^{j}_{source}}{||Q^{j}_{source}||_2} ||_2$

2.6. Learning What and Where to Transfer

La méthode Learning What and Where to Transfer (ou l2t-ww) a été introduite par Jang, Lee, Hwang et Shin[7] en 2019. Elle utilise des meta-réseaux pour apprendre quelles paires de couches du réseau source et du réseau cible devraient être connectées pour le transfert de connaissance, et quelles feature maps au sein de ces couches devraient être transférées et dans quelle quantité.

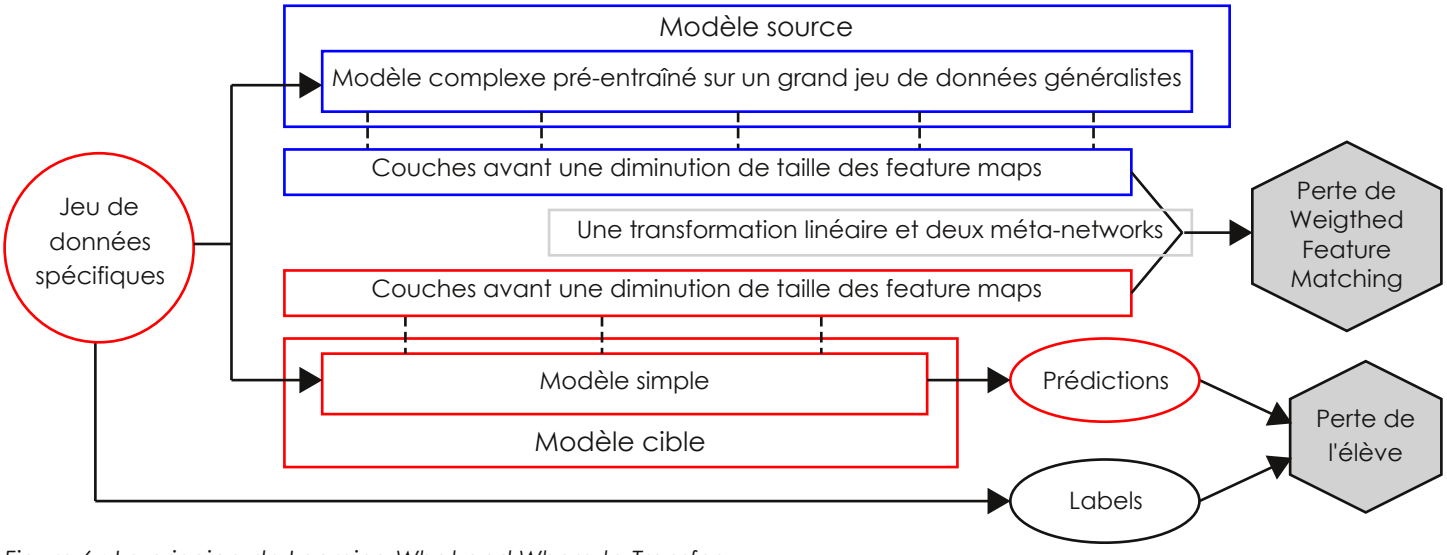


Figure 6 : Le principe de Learning What and Where to Transfer

7. Yunhun Jang, Hankook Lee, Sung Ju Hwang, Jinwoo Shin : Learning What and Where to Transfer, *International Conference on Machine Learning (ICML 2019)*, 2019.

Cette méthode essaie de réunir les avancées des différentes méthodes citées précédemment. Elle emprunte au Learing without Forgetting la possibilité de traiter une tâche différente avec le modèle cible. Elle utilise la notion de feature matching introduite par les FitNets. Elle adopte l'utilisation de plusieurs couches intermédiaires comme dans l'Attention Transfer.
Le réseau source est un réseau complexe pré-entraîné sur une tâche. Le réseau cible est un réseau plus simple qui cherche à apprendre une autre tâche.
Le réseau cible, ainsi que le transfert d'information du réseau source, est caractérisé par un set de paramètres θ.
Les méta-réseaux sont caractérisés par un set de paramètres ϕ.

Deux méta-networks permettent de savoir quoi transférer et où le transférer. Ils **se focalisent sur les couches qui précèdent une diminution de taille des feature maps**, aussi bien dans le modèle source que dans le modèle cible.
Le premier, nommé Weight Network dans le code et f dans le papier, détermine les feature maps du modèle source les plus importantes à transférer pour chaque couche. Ses entrées sont les feature maps du modèle source pour les couches sélectionnées. Il est constitué d'une couche dense qui a autant de sorties que d'entrées pour chaque paire de couches, et une fonction d'activation softmax. Ainsi, la somme des pondérations des feature maps transférées est égale à 1 pour chaque paire de couches. Tout comme dans les FitNets, une transformation linéaire opérée grâce à une convolution fait correspondre la taille des feature maps du réseau cible à la taille des feature maps du réseau source pour chaque paire de couches.
Le deuxième meta-network, nommé Loss Weight Network dans le code et g dans le papier, pondère la quantité d'informations transmises du réseau source vers le réseau cible pour chaque paire de couches. Ses entrées sont les feature maps du modèle source pour les couches sélectionnées. Il est composé d'une couche dense qui a une seule sortie pour chaque paire de couches, et une fonction d'activation ReLU6 afin d'assurer des poids positifs et pas trop élevés.
Cela donne les formules de pertes ci-dessous.

Perte de weighted feature matching entre deux couches m du réseau source et n du réseau target :

$$L_{wfm}^{m,n}(\theta|x,\omega^{m,n}) = \frac{1}{H \times W}\sum_{c}\omega_c^{m,n}\sum_{i,j}\left(r_\theta\left(T_\theta^n(x)\right)_{c,i,j} - S^m(x)_{c,i,j}\right)^2$$

avec S^m = couche m du réseau source
T_θ^n = couche n du réseau cible
r_θ = convolution avec un noyau de taille 1, le nombre de feature maps de T_θ^n en entrée et le nombre de feature maps de S^m en sortie = transformation linéaire pour ramener la dimension des feature maps de T_θ^n à celle des feature maps de S^m
H et W = hauteur et largeur des features maps de S^m
$\omega_c^{m,n}$ = importance de l'information contenue dans la feature map c de S^m pour T_θ^n (0 <= poids <= 1)
i va de 1 à H ; j va de 1 à W ; c va de 1 au nombre de feature maps de S^m
$\omega^{m,n} = [\omega_c^{m,n}]$ est la sortie du meta-network $f_\phi^{m,n}\left(S^m(x)\right)$
$\sum_c \omega_c^{m,n}$ = 1 car on applique une activation softmax à la sortie de f

Perte de weighted feature matching :

$$L_{wfm}(\theta|x,\phi) = \sum_{(m,n)}\lambda^{m,n}L_{wfm}^{m,n}(\theta|x,\omega^{m,n})$$

avec $\lambda^{m,n}$ = quantité d'information transférée de S^m vers T_θ^n (chaque valeure est comprise entre 0 et 6)
$\lambda^{m,n}$ est la sortie du meta-network $g_\phi^{m,n}\left(S^m(x)\right)$

Perte totale :

$$L_{total}(\theta|x,y,\phi) = Cross\text{-}Entropy(predictions,labels) + \beta \times L_{wfm}(\theta|x,\phi)$$

avec $Cross\text{-}Entropy(predictions,labels)$ = la cross entropy du modèle cible = perte originale
β = un hyperparamètre du modèle, qui permet de régler l'influence du Weighted Feature Matching

Equations 3 : Fonctions de pertes du modèle Learning What and Where To Transfer

L'approche l2t-ww nécessite d'entraîner deux meta-networks et la convolution de transformation linéaire pour diminuer la perte de Weighted Feature Matching, et le modèle cible pour diminuer la perte originale.
Pour cela, la méthode propose d'opérer en quatre temps pour chaque batch de données :
1. Calcul de la perte totale et des gradients qui lui sont liés. Mise à jour de θ (poids du modèle cible et de la transformation linéaire) pour minimiser cette perte totale.

2. Répéter 2 fois (T=2) : calcul de la perte de Weighted Feature Matching et des gradients associés, et mise à jour de θ.
3. Calcul de la perte originale et des gradients associés, et mise à jour de θ.
4. Calcul de la perte originale et des gradients associés, et mise à jour de ϕ (poids des meta-networks).

3. Méthode implémentée

3.1. Les objectifs

Je choisis d'implémenter la méthode Learning What and Where To Transfer pour répondre à plusieurs questions :
- Est-ce que je peux améliorer mon modèle CNN personnel très simple en transférant la connaissance de feature maps d'un modèle très complexe et récent comme EfficientNet B4 ?
- Est-ce que je peux améliorer l'accuracy d'un modèle plus ancien et simple comme VGG16 (avec une sortie Global Average Pooling pour diminuer le nombre de paramètres) en transférant la connaissance de feature maps d'un modèle plus complexe et récent comme EfficientNet B4 ?
- Est-ce que je peux améliorer un modèle EfficientNet plus simple (la version B0) avec un modèle EfficientNet plus complexe tel que la version B7 qui est la plus complexe dont le pré-entraînement est disponible avec Pytorch ?

3.2. Les données

J'utilise les données du Stanford Dogs Dataset. Pour rappel, ce jeu de données comporte 20580 images de chiens de 120 races différentes, issues de la base de données ImageNet.
Je répartis mes données en trois jeux de données, en respectant une stratification basée sur la race des chiens (le label de mon problème de classification) : entraînement (64%), validation (16%) et test (20%). Je base mes apprentissages sur un échantillon de 10% des données d'entraînement respectant la proportion des races, afin d'accélérer leur calcul.

3.3. Les outils

J'ai traité initialement ce problème classification multi-classe avec Tensorflow, avec une approche de Transfer Learning basée sur EfficientNetB4, avec le fine-tuning de la seule dernière couche de convolution. J'utilise dans ce projet des modèles pré-entrainés sans fine-tuning.
La méthode Learning What and Where To Transfer est codée par ses créateurs avec Pytorch. Je décide donc de travailler à la résolution de mes objectifs en Pytorch. Le code est initialement prévu pour des tâches pré-définies et les seuls modèles VGG et Resnet, et est appelable par ligne de commande. J'adapte le code pour généraliser les modèles utilisables et les tâches en objectif, et pour l'intégrer à ma méthode de travail dans des Notebooks.
Je recherche les modèles pré-entraînés qui m'intéressent. Certains sont disponibles sur Pytorch Vision. D'autres non. J'adapte le code à ma problématique, notamment en modifiant la fonction forward pour pouvoir obtenir l'extraction de feature maps intermédiaires en plus de la classification.
J'effectue les calculs avec l'accès au GPU offert par Google Colab.

3.4. Problèmes rencontrés : un nombre trop élevé de feature maps

La méthode utilise une convolution pour faire correspondre les feature maps de la couche du modèle cible à celles du modèle source. Il y a une convolution par paire de couches reliées entre le réseau source et le réseau cible. Chacune de ces convolutions est caractérisée par un nombre de canaux en entrée égal au nombre de feature maps du modèle cible, un nombre de canaux en sortie égal au nombre de feature maps du modèle source, et un noyau de taille 1x1.
Les couches extraites d'EfficientNetB4 correspondent aux sorties de blocs qui présentent une diminution de la taille des feature maps par rapport à la sortie du bloc précédent. Avec cette méthode de définition automatique des couches qui nous intéressent, j'extrais cinq couches d'EfficientNetB4, avec respectivement 24, 32, 56, 160 et 1792 feature maps.

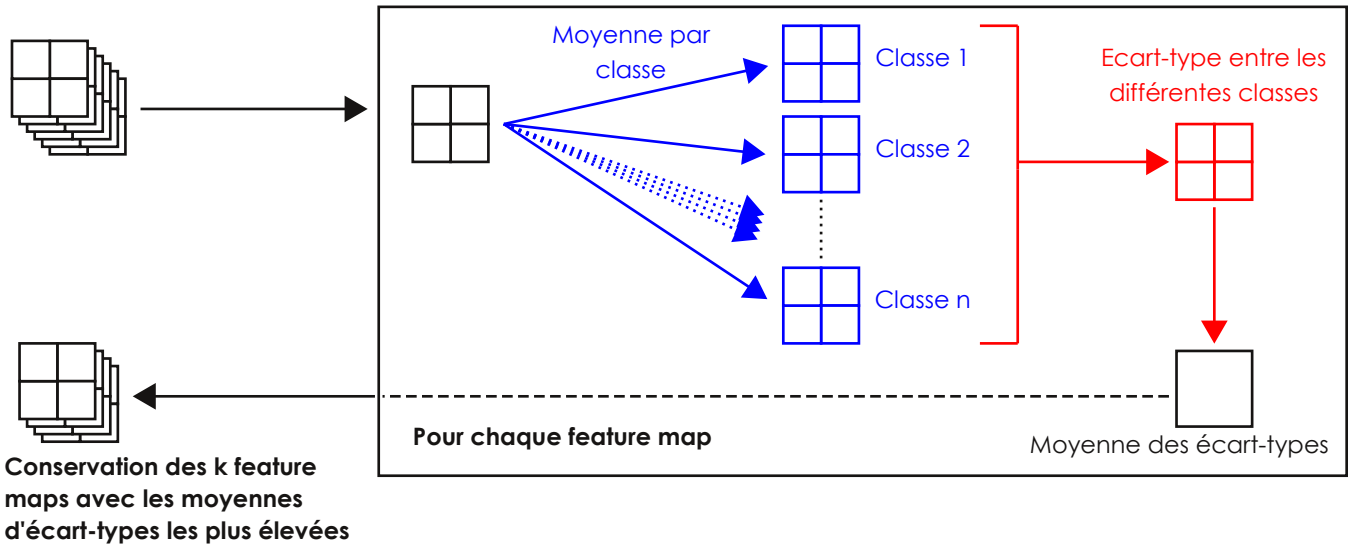


Figure 7 : Sélection des feature maps les plus discriminantes pour une couche donnée - k est fixé à 512

La création des convolutions pour les paires mettant en jeu cette dernière couche avec 1792 feature maps occupe trop de mémoire par rapport à ce que Google Colab alloue. Je dois donc trouver une astuce pour limiter ce nombre de feature maps, tout en gardant les feature maps qui m'offrent le plus de renseignements par rapport à ma tâche.
Je crée un module python qui gère l'extraction des features. Il détermine les feature maps qui discriminent le mieux les différentes classes de ma tâche. Puis il propose un filtre pour renvoyer les feature maps les plus discriminantes, en fonction d'un nombre maximum de features comme paramètre.

Je choisis une approche statistique pour déterminer les feature maps les plus discriminantes. Je pose l'hypothèse que les feature maps les plus discriminantes sont celles qui ont des activations les plus différentes d'un label (classe) à l'autre.
Je calcule donc les sorties pour toutes les couches du modèle à partir des données de mon problème. J'effectue ensuite les calculs suivants pour chaque couche dont je souhaite déterminer les feature maps les plus discriminantes :
- je calcule la moyenne des activations par label pour chaque pixel de chaque feature map ;
- je calcule ensuite l'écart-type de ces moyennes pour chaque pixel de chaque feature map. Plus les valeurs seront différentes entre les différents labels, plus cet écart-type sera élevé ;
- j'effectue la moyenne de ces écart-types par feature map;
- je ne conserve que les k feature maps avec la plus grande moyenne des écart-types, où k est le nombre pré-défini de feature maps à conserver.

Je choisis de ne garder que les 512 feature maps les plus discriminantes lorsqu'une couche extraite en contient plus. Ainsi, j'extrais cinq couches d'EfficientNetB4 avec respectivement 24, 32, 56, 160 et 512 feature maps.
Cette restriction du nombre de couches extraites résoud le problème de mémoire rencontré.

3.5. Problèmes rencontrés : une taille de batches trop grande

La méthode utilisée doit entraîner deux meta-networks et la convolution qui sert à la transformation linéaire, en plus du modèle cible. Par conséquent, elle nécessite une mémoire RAM importante.
Afin de tenir dans la mémoire allouée par Google Colab, je dois limiter la taille des batches à 8 images. Cette faible taille de batches implique plus de calculs pour chaque époque et donc un temps d'apprentissage assez long.

3.6. Problèmes rencontrés : l'explosion du gradient

Lors de mes premiers essais de transfert de connaissance d'EfficientNetB4 vers mon réseau très simple, je constate que le calcul de l'erreur pose problème. Je m'aperçois que l'origine du problème est une explosion de la perte, liée à une instabilité du gradient.
Afin de remédier à ce problème, j'ajoute une couche de batch normalization après chaque convolution dans mon réseau très simpe. Cet ajout permet de stabiliser le gradient et la perte du modèle, et ainsi de procéder à l'apprentissage du modèle.
Je rencontre le même problème lors du test du transfert de connaissance d'EfficientNetB4 vers un réseau VGG16. Pour rappel, ce réseau VGG16 ne comporte de couche de batch normalization. Utiliser le réseau VGG16_bn, qui est le même modèle que VGG16 avec une couche de batch normalization ajoutée après chaque convolution, résoud une nouvelle fois le problème.

4. Améliorer un réseau très simple avec un réseau très complexe

J'applique la méthode Learning What and Where To Transfer en choisissant comme réseau cible mon réseau CNN très simpe et comme réseau source un EfficientNetB4.
Mon réseau CNN très simple comporte trois couches de convolution avec une taille de noyau de 3x3, chacune suivie d'une batch normalization, d'une fonction d'activation ReLU, d'un spatial dropout de 20% et d'un max pooling avec une taille de noyau de 2x2. La sortie finale se fait avec un Global Max Pooling.

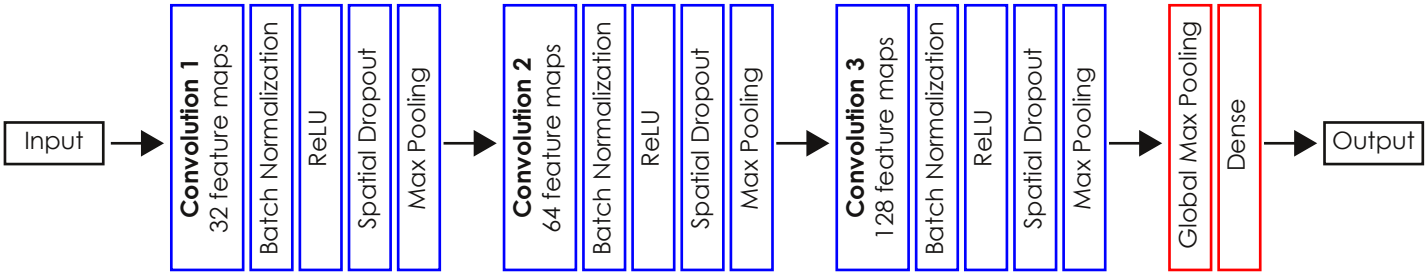


Figure 8 : Schéma du modèle CNN très simple

4.1. La performance Baseline

J'entraîne mon modèle simple avec l'optimiseur Adam, pour lequel je garde les paramètres par défaut excepté epsilon que je fixe à 1e-8 pour être en adéquation avec Tensorflow. La meilleure accuracy de validation obtenue est 5,59%, avec une perte de validation de 4,842. Ce modèle très simple est très peu performant.

4.2. La performance de la méthode l2t-ww

Je calcule la performance de la méthode en utilisant deux versions de mon réseau CNN comme modèle cible.
La première version se base sur un modèle dont les poids ne sont pas initialisés par ceux de notre entraînement préalable qui a permis d'obtenir la performance Baseline. La deuxième version utilise le modèle simple pré-entraîné.
Dans les deux cas, la performance est très proche et très faible. Elle est moins bonne que le modèle simple seul :
- sans pré-entraînement : accuracy de validation = 1,27% et perte de validation = 4,785.
- avec pré-entraînement : accuracy de validation =1,21% et perte de validation = 4,785.

4.3. Analyse de la contre-performance

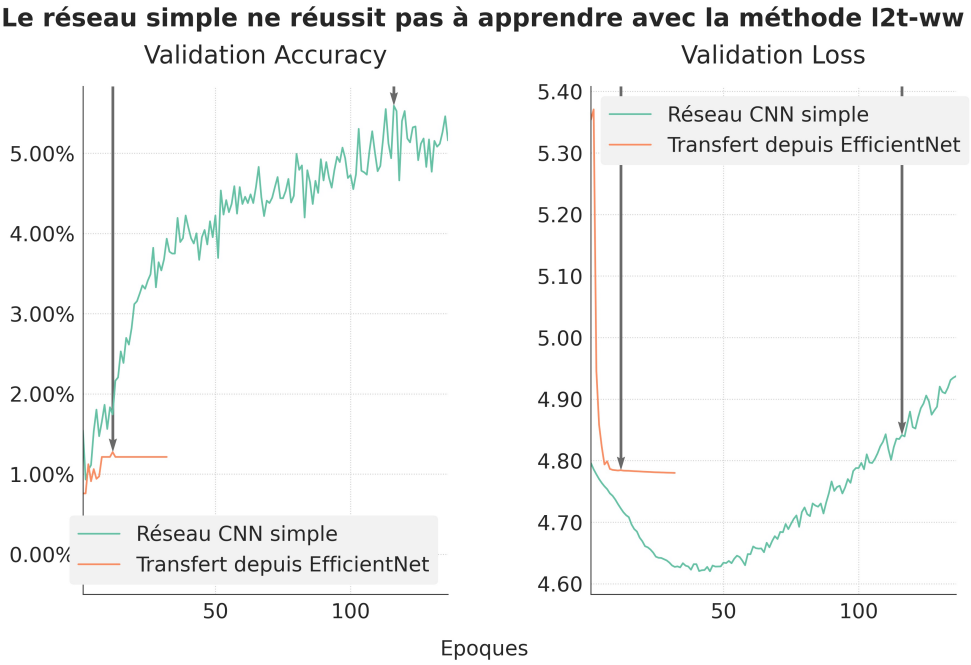


Figure 9 : Le CNN très simple ne réussit pas à apprendre avec la méthode l2t-ww

Notre réseau très simple n'arrive pas à apprendre des features extraites d'EfficientNetB4. Cela s'explique par la trop grande simplicité de notre modèle.

Rappelons nous que le feature matching correspond correspond à une sorte de régularisation. Son effet est dilué par les couches suivantes s'il est appliqué trop tôt dans un réseau cible profond. A l'inverse, les features extraites ne peuvent pas être correctement exploitées par le réseau cible si le feature matching intervient trop tard dans le réseau cible.

Ici, notre réseau cible est très simple et peu profond. Il est trop simple et pas assez profond pour exploiter les features transférées par knowledge distillation.
La méthode l2t-ww devrait présenter plus d'intérêt en transférant l'information vers le modèle VGG16.

4. Améliorer VGG16 grâce à EfficientNetB4

J'applique la méthode Learning What and Where To Transfer en choisissant comme réseau cible un VGG16 avec batch normalization et comme réseau source un EfficientNetB4.

4.1. La performance Baseline

Le modèle VGG16 avec batch normalization pré-entraîné sur ImageNet obtient une accuracy de validation de 66,93% sur notre problème de classification, pour une perte de validation de 1,669.
C'est bien meilleur que pour notre modèle très simple mais cela reste bien inférieur à l'accuracy de validation d'EfficientNetB4 qui dépasse 90%.

4.2. La performance de la méthode l2t-ww

L'apprentissage est très long. Je suis freiné par les ressources techniques à ma disposition. J'utilise l'entraînement sur GPU grâce à Google Colab, sur un GPU Tesla T4. La complexité de la méthode par rapport à la RAM à disposition me forcent à limiter la taille des batches à 8 images. L'entraînement du modèle est très long : je réussis à entraîner environ 40 époques par jour.
La lenteur de l'apprentissage m'amène à arrêter après 160 époques (5 jours de calculs), avant que le early stop ne se déclenche. Je ne sais pas jusqu'à quel niveau l'accuracy serait monté si l'apprentissage avait été à son terme. Mais je doute pouvoir obtenir d'aussi bonnes performances que celles présentées dans le papier.
En effet, l'apprentissage est très progressif, par petits pas, et la perte d'entraînement est déjà très basse (0,0188). Pour comparaison, la perte d'entraînement lors de l'optimisation du VGG16 avec le SGD Nesterov est de 0,919 après 200 époques. Le potentiel d'apprentissage résiduel paraît donc limité sur la méthode l2t-ww après 160 époques.
Après ces 160 époques, je n'ai atteint qu'une accuracy de validation de 16,11%, pour une perte de validation de 4,318.

4.3. L'analyse de l'apprentissage de la méthode l2t-ww

Plusieurs facteurs peuvent entraver le bon apprentissage du modèle.
D'une part, je n'utilise que 10% du jeu de données d'entraînement pour l'apprentissage afin de conserver une durée

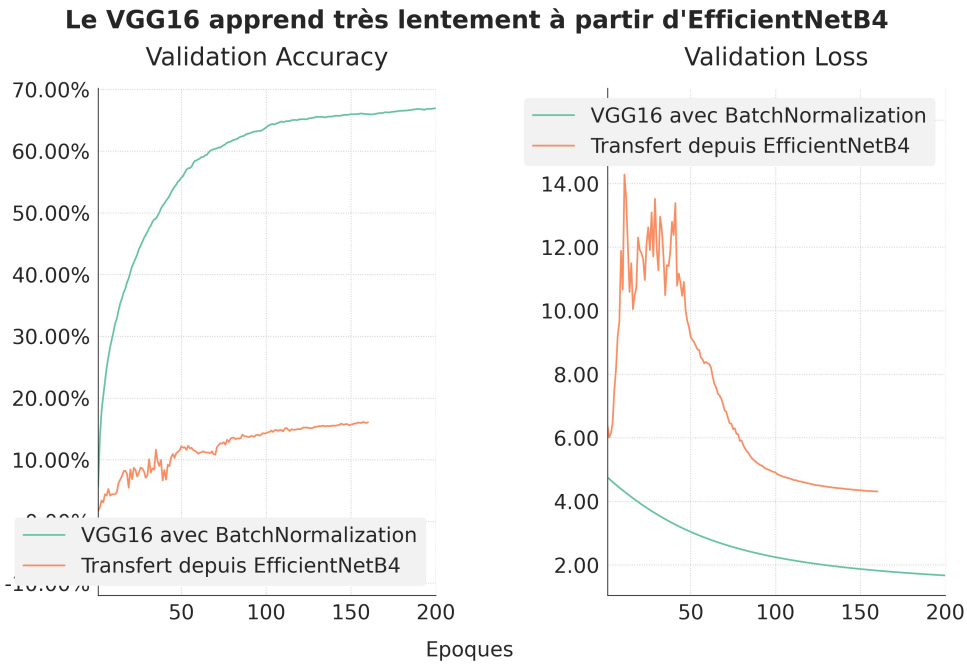


Figure 10 : L'apprentissage avec la méthode l2t-ww est très lent

raisonnable et compatible avec les délais de mon projet. La méthode peut potentiellement apprendre de façon plus performante avec plus de données.
Il est intéressant de noter ici que l'utilisation de seulement 10% des données n'avait pas freiné l'apprentissage avec l'approche du Tranfer Learning. L'accuracy obtenue avec 10% des données et toutes les données étaient très proches.
D'autre part, j'ai fait tourner la méthode sans data augmentation. Même si la data augmentation permet en général d'améliorer l'apprentissage des modèles, son absence ne peut pas expliquer à elle seule le manque de performance observé sur nos approches.

Enfin, j'ai dû limiter le nombre de feature maps transférées par la dernière couche d'EfficientNet B4 à 512 channels (sur les 1792 feature maps disponibles), pour respecter les limites de RAM à ma disposition. Bien que j'ai développé une stratégie pour transférer les 512 feature maps les plus discriminantes pour notre tâche, je ne peux pas exclure que la conservation de seulement 28,6% de l'information de la dernière couche d'EfficientNetB4 prive le modèle d'informations importantes.

Ces réserves sont toutes liées à un même élément. La méthode est très complexe. Elle cumule plusieurs réseaux et comprend beaucoup de paramètres, ce qui la rend très gourmande en ressource. Pour cette même raison, elle est sans doute propice à l'overfitting, et ce d'autant plus que le jeu de données est petit.

4. Améliorer EfficientNetB0 grâce à EfficientNetB7

J'applique la méthode Learning What and Where To Transfer en choisissant comme réseau cible un EfficientNetB0 et comme réseau source un EfficientNetB7. L'idée est de tester la méthode avec deux réseaux qui ont une structure semblable, pour voir si le transfert d'information est plus efficace dans cette configuration.

4.1. La performance Baseline

Le modèle EfficientNetB0 pré-entraîné sur ImageNet obtient une accuracy de validation de 57,89% sur notre problème de classification, pour une perte de validation de 2,260. Cette perte de validation est assez élevée, tout comme la perte d'entraînement qui s'établit à 1,826. Cela laisse présager d'un potentiel d'amélioration sensible de la performance en appliquant un fine-tuning.
Il est intéressant de noter que le modèle EfficientNetB0 est particulièrement léger, avec "seulement" 5,33 millions de paramètres.

4.2. La performance de la méthode l2t-ww

La méthode l2t-ww ne parvient pas à apprendre. Elle fait face à une instabilité du gradient, malgré la présence de couches de batch normalization dans les réseaux EfficientNet. La méthode paraît donc inadaptée lorsque l'on choisit un réseau EfficientNet comme réseau cible.
Cela pose le problème du caractère universel de cette méthode, qui ne paraît pas s'adapter correctement à tous les types de réseaux.

5. Conclusions de l'étude

La méthode Learning What and Where To Transfer m'a paru très prometteuse à la lecture du papier. Je n'ai pas pu mener mes explorations aussi librement que souhaité et nécessaire du fait de mes limitations en ressource matérielle. C'est le premier point négatif de cette méthode : elle est **très gourmande en ressources**.
Le fait qu'elle implique un grand nombre de paramètres me fait également craindre un **risque d'overfitting**.
Enfin, les auteurs posent une hypothèse d'universalité, évoquant la possibilité d'utiliser leur méthode sur d'autres types de réseaux que les CNN. **Je doute fortement de cette possibilité de généralisation** à tous les réseaux, de fait qu'elle n'est déjà pas adaptable à tous les types de réseaux CNN.
Je pense que la performance de cette méthode est étroitement liée à la sélection des modèles cible et source.

6. Références

1. Stanford Dogs Dataset : 20580 images de chiens de 120 races différentes, issues de la base de données ImageNet, disponibles sur le site de Stanford http://vision.stanford.edu/aditya86/ImageNetDogs/main.html

2. https://towardsdatascience.com/transfer-learning-from-pre-trained-models-f2393f124751

3. Geoffrey Hinton, Oriol Vinyals, Jeff Dean : Distilling the Knowledge in a Neural Network, *Deep Learning and Representation Learning Workshop, Advances in Neural Information Processing Systems 29 (NIPS 2015)*, 2015.

4. Zhizhong Li, Derek Hoiem : Learning wihout Forgetting, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2018.

5. Adriana Romero, Nicolas Ballas, Samira Ebrahimi Kahou, Antoine Chassang, Carlo Gatta, Yoshua Bengio : Fitnets, Hints for Thin Deep Nets, *The 3rd International Conference on Learning Representations* (ICLR 2015), 2015.

6. Sergey Zaogoruyko, Nikos Komodakis : Paying More Attention To Attention: Improving the Performance of Convolutional Neural Networks via Attention Transfer, *The 5th International Conference on Learning Representations* (ICLR 2017), 2017.

7. Yunhun Jang, Hankook Lee, Sung Ju Hwang, Jinwoo Shin : Learning What and Where to Transfer, *International Conference on Machine Learning (ICML 2019)*, 2019.

7. Index des illustrations