Formation Machine Learning

PROJET 7

Développez une preuve de concept

Février 2021

Maud Charbonneau

Sommaire

Introduction

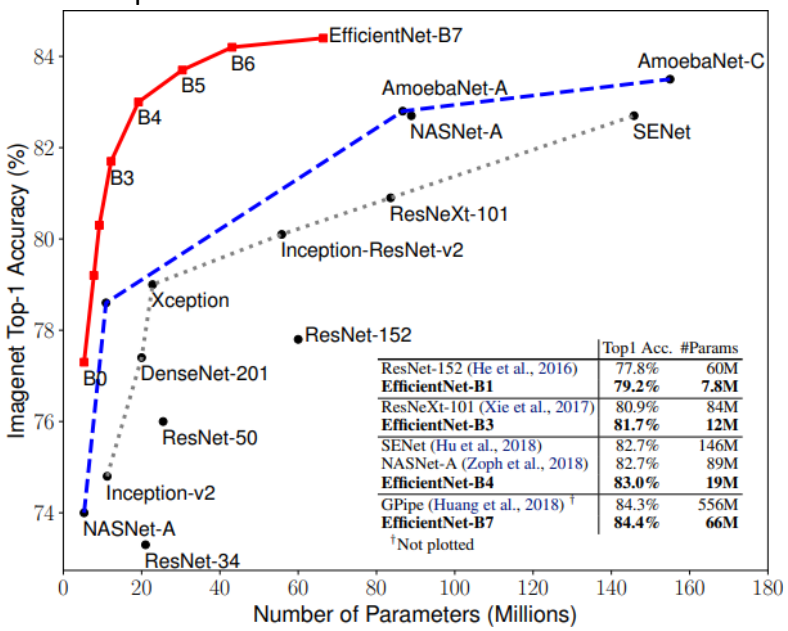
Ces dernières décennies, le monde de l'informatique a évolué rapidement. En Machine Learning il est donc important de savoir progresser en même temps. Il faut être capable de rechercher des nouveaux outils plus performants et de savoir les implémenter. Ce projet permet donc de s'entrainer.

Dans le cadre de ce projet, j'ai décidé de me baser sur un projet précédent : le projet 6 - "Classez des images à l'aide d'algorithmes de Deep Learning". Ce projet avait pour but de trouver un algorithme performant pour classifier des images de chiens par race.

Le but est donc d'effectuer des recherches afin de trouver un algorithme récent plus performant et de l'implémenter afin d'obtenir un meilleur résultat que celui obtenu lors du projet 6. Pour cela, le dataset ainsi que la méthode de baseline sont ceux du projet 6 et seront rappelés par la suite dans ce rapport.

Etat de l'art

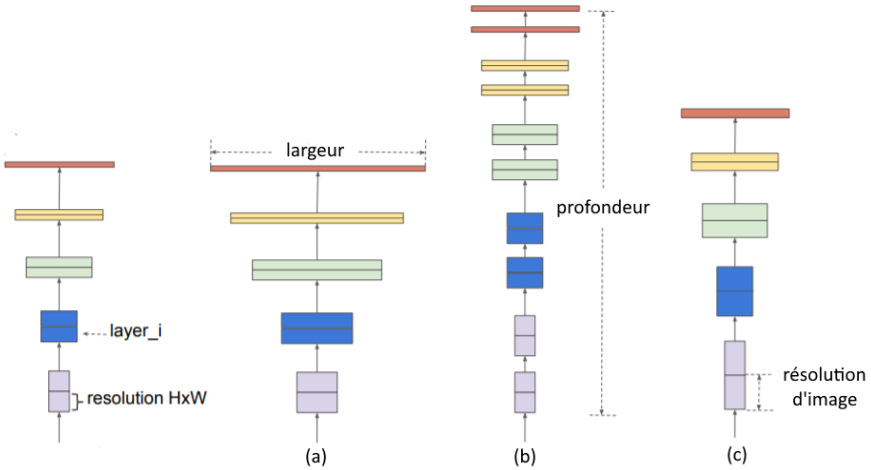
La méthode utilisée dans ce projet est *EfficientNet*. Ce groupe de réseaux de neurones convolutifs a été présenté la première fois à l'*International Conference on Machine Learning* en juin 2019. Il comporte huit modèles allant de *B0* à *B7* ayant différents nombres de paramètres et atteignant des précisions plus importantes que ses prédécesseurs tout en ayant beaucoup moins de paramètres.



Performances de différents réseaux de neurones convolutifs

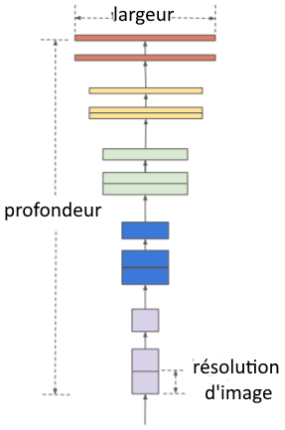
La différence entre ce groupe de réseaux et les autres se trouve dans la méthode d'optimisation. Sur les réseaux avant *EfficientNet*, pour optimiser un modèle il était possible de modifier soit la profondeur du réseau, soit sa largeur, soit la résolution de l'image. La majorité des réseaux ne joue que sur une des dimensions. Il est cependant possible d'optimiser deux ou trois dimensions de façon arbitraire impliquant donc beaucoup de réglages fastidieux. Cela mène souvent à des résultats peu satisfaisants.

Ce qui fait la particularité de *EfficientNet* est sa méthode d'optimisation : le *compound method scaling* qui optimise les trois dimensions à la fois. Le principe est d'optimiser chaque dimension de façon proportionnelle au lieu de le faire de façon arbitraire.



Architecture d'un réseau de base et dimensions mises à l'échelle - (a) : optimisation de la largeur ; (b) optimisation de la profondeur ; (c) : optimisation de la résolution d'image

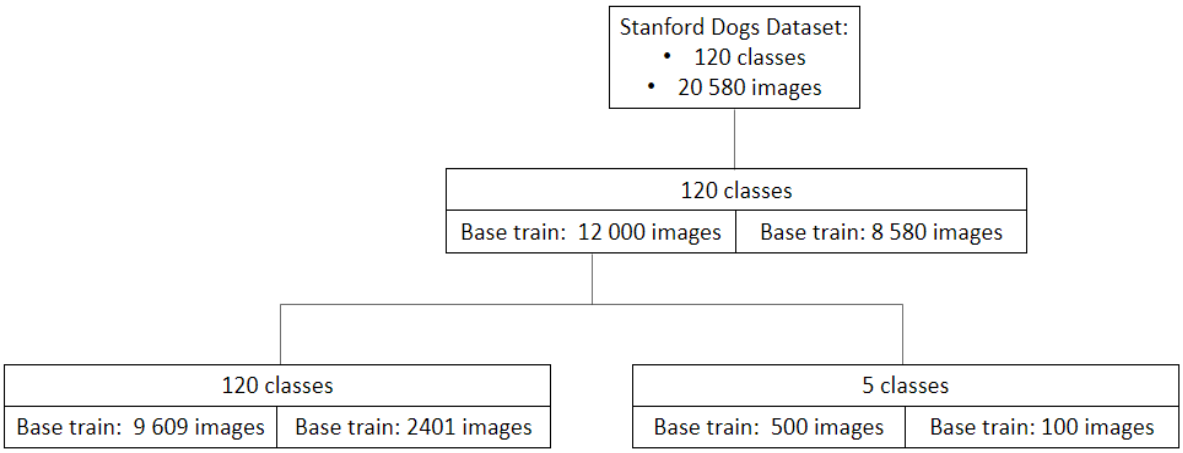
Pour reprendre l'exemple employé dans l'article de recherches : si on choisit d'utiliser 2 N fois plus de ressources de calcul, le réseau est augmenté en profondeur de α^N, en largeur de β^N, et la résolution d'image par γ^N, où α, β, γ sont des coefficients constants déterminés. Ces coefficients sont trouvés en se référant à un tableau de données défini grâce à un modèle plus petit.



Architecture d'un réseau EfficientNet optimisé avec la méthode de compound scaling

Jeu de données

Pour ce projet, je me suis appuyée sur le projet 6 au niveau du dataset et de la méthode baseline. Le but est donc d'arriver à un meilleur résultat que le projet 6. Pour rappel, le projet 6 a pour but de classifier des images de chiens en fonction de leur race. Le jeu de données utilisé est le Stanford Dogs Dataset. Il comprend 20 580 images correspondant à 120 races.



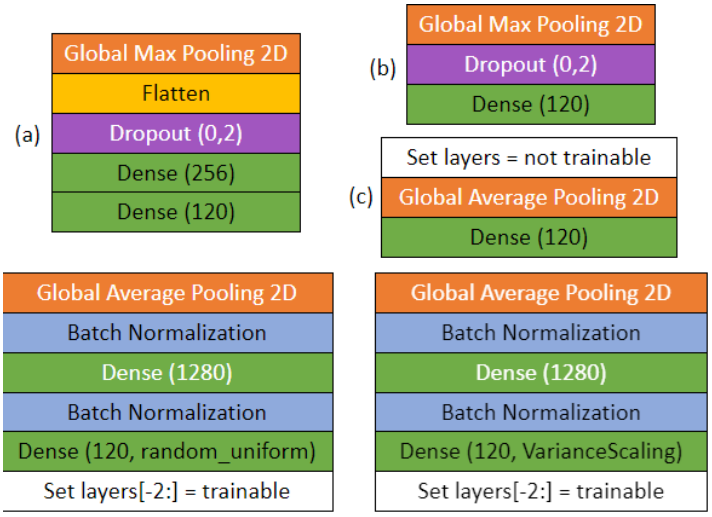
Séparation du jeu de données

Pour tester le nouveau modèle, le dataset est séparé en une partie train et test. Cependant, pour tester différents modèles, une autre base de données est créée à partir de la base train. C'est une base, contenant 120 classes et 12 000 images, séparée en train et test. Dans le précédent projet, une autre base était également utilisée pour tester plus de paramètres plus rapidement. Cette base contenait uniquement 600 photos choisies aléatoirement dans la base train originale et ne contenait que 5 classes. Dans le cadre de ce projet, cette base n'est pas aussi utile car, puisque le but est d'optimiser le modèle, la base est trop petite pour être suffisamment précise.

EfficientNet: implémentation et performances

Implémentation et Transfert Learning

L'implémentation de *EfficientNet* est similaire aux autres modèles que j'ai pu tester précédemment puisque c'est un modèle qui utilise Keras. Keras est une bibliothèque open-source écrite en Python qui est très employée en Deep Learning. Il suffit donc d'importer *EfficientNet*. L'utilisation du Transfert Learning sur ce modèle va ensuite permettre de l'ajuster au sujet. J'ai voulu tester différentes architectures pour les dernières couches du réseau. J'ai donc testé quatre modèles différents nommés : base, base simplifiée provenant de différents articles, TL-proj6 étant la même architecture de Transfert Learning que celle utilisée pour la baseline et deux modèles de fine tuning extrait de Kaggle ayant différents paramétrages. Le premier modèle de fine tuning nommé RU a comme paramètre : kernel_initializer = random_uniform. Le deuxième modèle nommé VS a pour initialiseur VarianceScaling. Pour ne pas entrer dans le détail de l'architecture de ces modèles, ils seront tous récapitulés dans cinq tableaux contenant les couches utilisées.

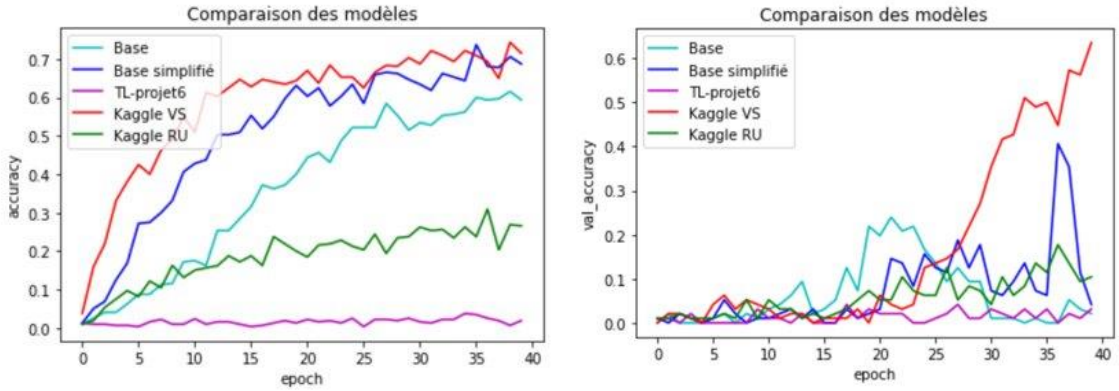


Architectures des dernières couches du modèle : (a) : Modèle base ; (b) : Modèle base simplifiée ; (c) : Modèle TL-Proj6 ; (d) : Modèle Kaggle Random Uniform ; (e) : Modèle Kaggle Variance Scaling

J'ai donc créé une fonction pour chaque type d'architecture et appliqué sur les modèles *EfficientNet-B0, EfficientNet-B1, EfficientNet-B2, EfficientNet-B3* et *EfficientNet-B4* puis en fonction des différents résultats obtenus en variant les paramètres, le meilleur modèle sera sélectionné.

Modèle	Optimiseurs	Learning rate	Précision train	Précision validation	Temps en s.
Base	Adam	0,0001	0.0968	0.0104	4007
Base simplifiée	Adam	0,0001	0.4749	0.0104	835
TL-projet6	Adam	0,0001	0.0197	0.0208	227
Kaggle VS	Adam	0,0001	0.6814	0.1146	209
Kaggle RU	Adam	0,0001	0.8303	0.1979	193

Précision de différentes implémentations de EfficientNet-B0



Courbes d'apprentissage des différentes implémentations de EfficientNet-B0

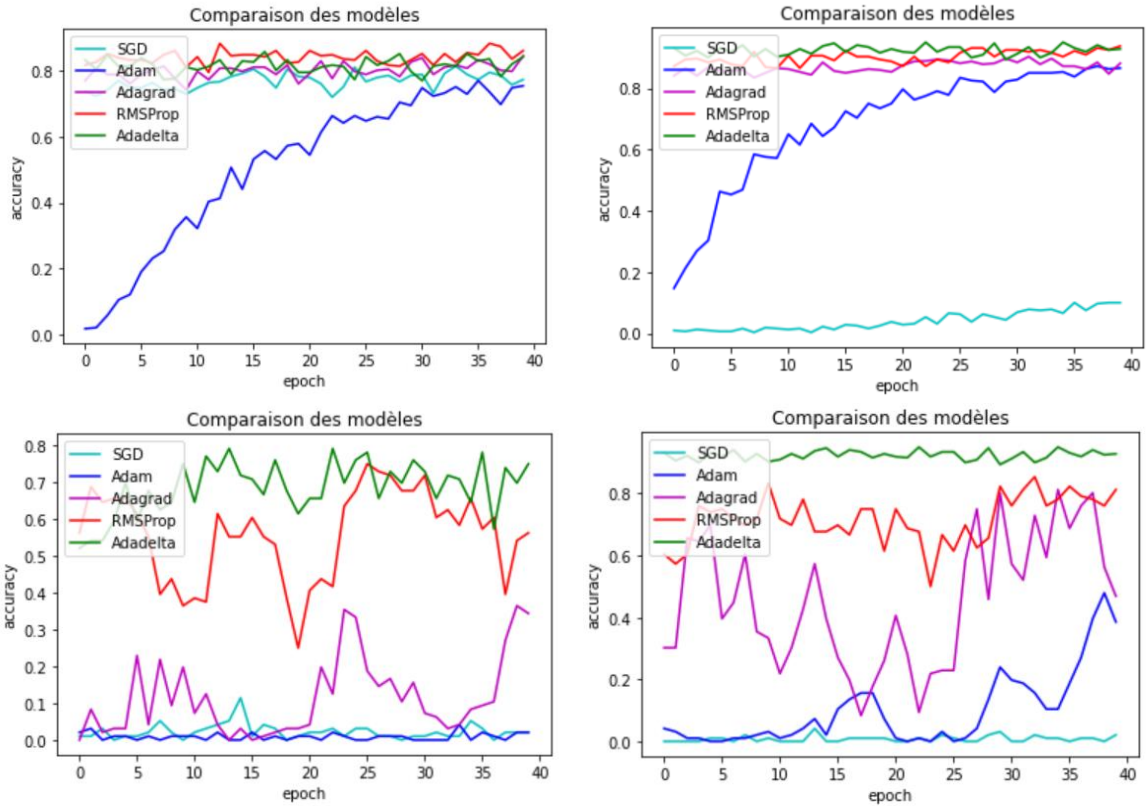
Voici les résultats pour *EfficientNet-B0* pour 40 epochs et des batchs de 40. Les modèles importés et modifiés de Kaggle ont de meilleurs résultats en précision train et validation aussi bien qu'au niveau du temps. Les tests effectués par la suite seront adaptés à ces modèles pour tenter de les optimiser. Le paramétrage, excepté le callback, a été effectué sur Kaggle

Random Uniform. C'est à cause de résultats non concluants du début du paramétrage que Kaggle Variance Scaling a été ajouté et testé avec certains paramètres.

Paramétrage

Optimiseurs

Pour tenter de trouver les meilleurs paramètres pour optimiser le modèle, j'ai testé cinq optimiseurs : *SGD, Adam, Adagrad, Adadelta* et *RMSprop*. Chacun a été utilisé avec les modèles de *B0* à *B4*.



Courbes d'apprentissage de précision et perte des différents optimiseurs de EfficientNet-B0 et EfficientNet-B3

Les courbes pour *B0* et *B3* ont été regroupées sur un graphique de la précision et de la perte contenant tous les optimiseurs. *SGD* a des résultats nettement inférieurs aux autres optimiseurs. *Adagrad* a une meilleure performance que ce dernier mais les résultats restent inférieurs à ceux de *Adadelta* et *RMSProp*. Ces deux optimiseurs donnent une bonne précision mais celle-ci stagne comme le montrent les courbes. *Adam* est le seul qui parait apprendre au fur et à mesure. Le problème est que la précision sur les valeurs de validations est très faible. Ces trois optimiseurs ont obtenu des meilleurs résultats sur tous les modèles et sont donc gardés pour la suite du paramétrage. Avant de commencer le fine-tuning, la meilleure précision obtenue était 69,89%.

Ces optimiseurs ont également été testés sur l'implémentation Kaggle Variance Scaling. On remarque que les résultats sont largement meilleurs même si la différence de précision entre train et validation est importante.

Modèle	Optimiseurs	Learning rate	Précision train	Précision validation	Temps en s.
EficientNet-B4	SGD	0.0001	0,0350	0,0104	453
EficientNet-B4	Adam	0.0001	0,8134	0,0417	456
EficientNet-B4	Adagrad	0.001	0,8798	0,7396	452
EficientNet-B4	RMSprop	0,00002	0,9002	0,6042	466
EficientNet-B4	Adadelta	0,001	0,8899	0,7708	455

Résultat de EfficientNet-B4 avec les différents optimiseurs pour 40 epochs et 40 images par batch

Nombre d'epochs et taille de batch

J'ai testé différents nombres d'epochs - 40, 50, 80, 100 - et différentes tailles de batch – 40 et 80. Pour ces variables, tous les modèles de B0 à B4 ont été testés et les conclusions de ces tests sont les suivantes :

- Les modèles *EfficientNet-B3* et *EfficientNet-B4* performent mieux que les autres
- Les meilleurs résultats sont obtenus pour 80 epochs et 40 images par batch pour EfficientNet-B3, et 80 epochs et 80 images par batch pour EfficientNet-B4.

Learning rate

Pour finir avec le fine-tuning, le dernier paramètre que j'ai testé est le learning rate. Pour cela j'ai testé quatre valeurs : 1e-2, 1e-3, 1e-4, 1e-6 sur *B3* uniquement avec les optimiseurs *Adadelta* et *RMSprop*. La meilleure précision lors de ces tests est avec *Adam* avec un learning rate de 0,0001.

Callbacks

Afin d'améliorer les performances du modèle, le callback *ReduceLROnPlateau* est utilisé. Ce paramètre permet de réduire le learning rate si l'une des métriques ne s'améliore plus. Lors des nombreux tests, un des problèmes récurrents était que la précision de validation ne s'améliorait pas suffisamment par rapport à la précision de train. L'ajout de ce callback a permis de nettement améliorer les résultats.

Comparaison avec la baseline

Tout comme le jeu de données, le modèle de baseline provient également du précédent projet. C'était le modèle ayant la meilleure précision : 85,94% sur la base train originale et 79,33% sur la base test originale. Pour rappel, le modèle était issu de *InceptionV3* avec un paramétrage regroupé dans le tableau suivant.

Modèle	InceptionV3
Stratégie de Transfert Learning	Extraction de features
Optimiseur	Adam
Learning rate	0,0001
Nombre d'epochs	40
Taille du batch	40

Paramètres du modèle de baseline

Analyse des résultats

Pour commencer, il a fallu tester les différents modèles de B0 à B4 pour voir la différence. Pour cela j'ai effectué les tests avec tous les optimiseurs mais les résultats affichés sont obtenus avec Adam. EfficientNet-B3 semble être le meilleur modèle au niveau de la précision. EfficientNet-B4 obtient le deuxième meilleur résultat mais celui-ci est très proche des autres modèles. De plus on observe que la précision de validation est très faible. Ces valeurs s'expliquent par l'implémentation du modèle n'étant pas encore optimisé au maximum. Par la suite, la modification du modèle ainsi que le callback évoqué précédemment permettent d'obtenir de très bons résultats.

Modèle	Optimiseurs	Learning rate	Précision train	Précision validation	Temps en s.
EficientNet-B0	Adam	0.0001	0,2705	0,0312	207
EficientNet-B1	Adam	0.0001	0,3304	0,0208	287
EficientNet-B2	Adam	0.0001	0,3285	0,0104	299
EficientNet-B3	Adam	0.0001	0,5286	0,0208	368
EficientNet-B4	Adam	0.0001	0,3573	0,0104	463

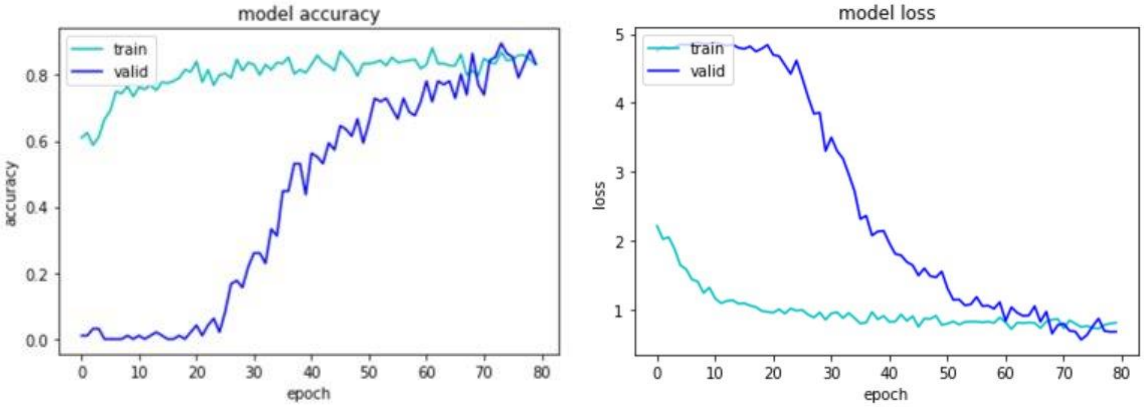
Précision des différents modèles de EfficientNet avec Adam, 40 epochs et 40 de batch size

Afin d'améliorer la précision de train et de validation il a fallu faire jouer les différents paramètres cités précédemment. Bien que certains optimiseurs aient de meilleurs résultats, ils ne semblaient pas apprendre au fur et à mesure. Pour cette raison j'ai décidé de me concentrer sur Adam en modifiant le nombre d'epochs et d'images par batch mais aussi en ajoutant le callback *ReduceLROnPlateau* qui permet de modifier le learning rate et qui a permis d'augmenter les précisions.

Modèle	Epoch	Batch size	Précision train	Précision validation
EfficientNet-B3	80	40	0,9132	0,8281
EfficientNet-B3	100	80	0,8717	0,8906
EfficientNet-B4	100	40	0,8511	0,8750
EfficientNet-B4	100	80	0,8312	0,8750

Résultats des meilleurs modèles avec Adam avec nombre d'epochs et taille de batch

Après différents tests, on obtient le meilleur modèle : *EfficientNet-B3* pour sur un batch size de 80 avec comme précision 85,04%.



Courbes d'apprentissage du meilleur modèle de EfficientNet-B3

Conclusion – Améliorations possibles

- Les résultats obtenus à travers ce projet montrent que la précision de *EfficientNet* est en effet meilleure que celle du modèle de baseline. Cependant les courbes d'apprentissage du meilleur modèle trouvé ne sont pas lisses ce qui montre un manque de stabilité de la précision. De nouvelles modifications au niveau du fine tuning ou du paramétrage pourraient apporter de la stabilité pour rendre le modèle plus performant.
- Il y a plusieurs modifications qui pourraient être appliquées pour améliorer ces modèles. Une modification possible serait d'augmenter linéairement la valeur de la couche dropout comme décrit dans l'article de EfficientNet. Une autre option serait de définir *drop_connect_rate* ce qui contrôle la valeur du dropout. Une autre modification serait de garder gelée la couche BatchNormalization, selon certains articles cela permettrait d'améliorer les performances. Enfin, un batch size faible permet d'accroitre la précision de validation, possiblement du a une régularisation plus efficace.
- Le modèle *EfficientNet-B5* ainsi que les suivants pourraient également être plus précis. Mais ces modèles sont plus importants car ils possèdent un très grand nombre de paramètres ce qui m'a empêchée de pouvoir les tester. Cependant avec certaines autres ressources, il serait possible de tester les autres modèles et de les paramétrer afin d'atteindre une précision encore meilleure.

Ressources

https://arxiv.org/abs/1905.11946
https://github.com/qubvel/efficientnet/blob/master/examples/inference_example.ipynb
https://github.com/Tony607/efficientnet_keras_transfer_learning/blob/master/Keras_efficientnet_transfer_learning.ipynb
https://keras.io/examples/vision/image_classification_efficientnet_fine_tuning/
https://amaarora.github.io/2020/08/13/efficientnet.html
https://colab.research.google.com/drive/1vzEDAX-3ol7gcZ7qmKuwn8zUld524sUZ#scrollTo=D9MM2WA49zyj
https://blog.roboflow.com/how-to-train-efficientnet/
https://colab.research.google.com/drive/1vzEDAX-3ol7gcZ7qmKuwn8zUld524sUZ#scrollTo=G7Evp4GOUt7Z
https://www.kaggle.com/devang/transfer-learning-with-keras-and-efficientnets
https://www.dlology.com/blog/transfer-learning-with-efficientnet/
https://heartbeat.fritz.ai/reviewing-efficientnet-increasing-the-accuracy-and-robustness-of-cnns-6aaf411fc81d