Plant Pathology 2020

Youssef Allouah, Paul Calot

Sommaire

1 Introduction

1.1 Préalable

Ce projet repose sur une publication scientifique et une compétition Kaggle s'étant déroulée du 9 mars au 26 mai 2020 [1], [2] .

1.2 Description du problème

Le mauvais diagnostic de maladies se propageant dans les champs peut mener à une mauvaise utilisation de produits chimiques ayant pour conséquence :

- Une plus grande résistance des pathogènes ;
- Une augmentation des coûts de production ;
- Diminution de la rentabilité économique ;
- Une destruction de l'environnement.

La détection par des humains de ces maladies consume beaucoup de temps et de moyens. De plus, une approche par vision par ordinateur avec des modèles de base n'est pas suffisante pour tenir compte de la grande variance des feuilles (âge, génétiques, symptômes etc.).

Dans le cas précis qui nous intéresse, il s'agit de développer une solution résiliante à un problème de classification à quatre classes : saine (*healthy*), multiple maladies (*multiple diseases*), rouille (*rust*) et tavelure (*scab*). La figure 1 présente un exemplaire de chaque classe.

Initialement, 1821 images de haute résolution ($2048 * 1365$) sont disponibles pour l'entraînement. La répartition de ces images entre les différentes classes est donnée par la table 1.

1.3 Idée générale de la résolution

Le jeu de données d'entraînement de 1821 images présente une certaine variété au sein d'un même type de feuilles : les symptômes peuvent varier d'une feuille à l'autre, l'âge crée des différences également ainsi que l'exposition au soleil, la présence de flou ou celle d'autres feuilles etc. Cependant, les modèles de base de réseau de neurones ne peuvent tenir compte facilement de cette diversité.

La précision des détails qui doivent être reconnus par le réseau de neurones impose de prendre une architecture plutôt profonde. Cependant, la faible quantité de données (1821 images) nous empêche d'entraîner trop de nouveaux poids ce qui met à mal la découverte des *features* pertinentes pour la classification.

Notre résolution présente donc principalement les blocs suivants :

- Augmentation du jeu de données ;
- Réutilisation d'architectures pré-entraînées et comparaison.

(a) Saine (b) Multiple maladies (c) Rouille (d) Tavelure

Figure 1: Exemplaires de chaque classe

Sain	Maladies multiples	Rouille	Tavelure
517	92	622	590

Table 1: Répartition initiale des classes.

2 Description de l'implémentation

2.1 Augmentation du jeu de données

2.1.1 Justification et résultat de l'augmentation des données.

Afin de justifier la nécessité d'augmenter les données, intéressons nous à l'architecture ResNet-50 pré-entraînée et à ses performances sur le jeu de données initial. Lorsque toutes les couches sont gelées et que l'on rajoute uniquement une couche dense permettant de passer de 1000 classes à seulement 4, on obtient les résultats de la figure 2. A noter que 60 % des données ont été utilisé pour l'entraînement.

On voit ainsi apparaître une limitation de la précision ainsi qu'un début de sur-apprentissage. Pour contrer cela, une augmentation du jeu de données a été réalisée avec la création d'une quinzaine de nouveaux jeux de données dont les meilleurs sont présentés sur la table 2. Chacune des images de ces jeux est à la résolution $224 * 224$.

Notons que le faible pourcentage de données pour la classe "maladies multiples" a abouti à la création de jeux de données additionnels rétablissant un certain équilibre. La matrice de confusion de la figure 2 montre un pourcentage de vrais positifs pour la classe 1 bien plus faible que dans le jeu de données d'évaluation. Cette différence est moins visible dans le jeu d'entraînement probablement à cause du sur-apprentissage.

Ces jeux de données ont été créés de la façon suivante :

1. Séparation entre le jeu d'entraînement et le jeu d'évaluation ;
2. Augmentation du jeu d'entraînement.

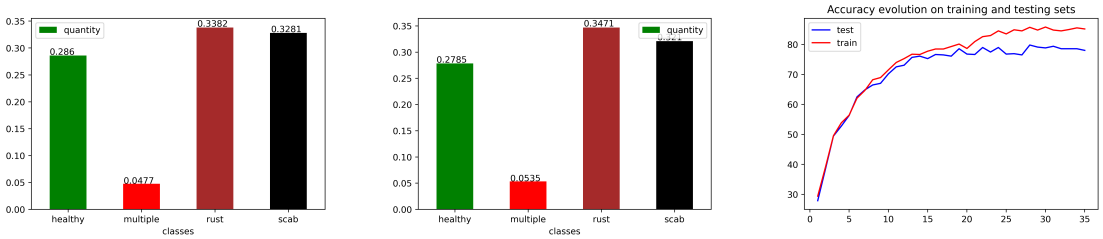
L'utilisation de ces jeux de données, ainsi que leurs forces et leurs faiblesses, seront abordées dans la partie suivante.

Un exemple d'augmentation sur une feuille de la classe *multiple maladies* est donné en figure 3.

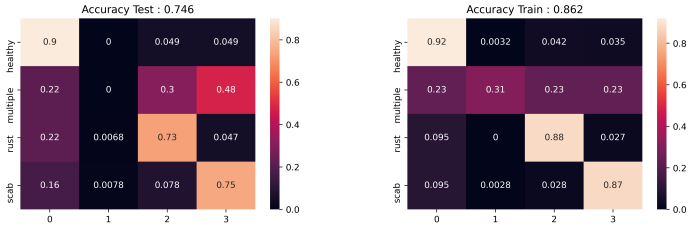
2.1.2 Difficultés rencontrées

Les difficultés que nous avons rencontrées dans l'établissement de ces jeux de données sont multiples :

1. Nous avions initialement augmenté tout le jeu de données avant de séparer entre données d'entraînement et d'évaluation. Dans ce cas de figure, deux images issus de la même image d'origine peuvent se retrouver dans le jeu d'entraînement pour la première et le jeu d'évaluation pour la seconde. Cela biaise la précision lors de l'évaluation et peut laisser croire à un bon modèle alors qu'il y a un sur-apprentissage massif ;



(a) Répartition jeu d'entraînement (b) Répartition jeu d'évaluation (c) Précision



(d) Matrice de confusion - jeu d'évaluation (e) Matrice de confusion - jeu d'entraînement

Figure 2: Résultats sur *ResNet-50* avec toutes les couches gelées et pour 35 epochs. Pour la matrice de confusion, les vraies classes sont notées en ordonnées, la somme sur une ligne des probabilités fait 1.

Jeu	aléatoire	ops. images	bruit	ops. pixels	% séparation	$\alpha_{1,3,4}$	α_2	total
0					60	1	1	1092
7	x	x	x	x	60	1	1	1092
5	x	x	x	x	77	1	6	1740
6	x	x	x	x	60	2	3	2238
4	x	x	x	x	77	2	10	3344
8	x	x	x	x	77	4	7	5804
10	x	x	x	x	77	4	12	6144
9	x	x	x	x	77	6	10	8672
11	x	x	x	x	77	6	16	9080
3classes	x	x	x	x	77	3	0	3994

Table 2: Augmentations réalisées. Aléatoire qualifie le caractère aléatoire de ces changements sur le jeu de données. Les trois colonnes suivantes qualifient dans l'ordre des modification de l'image (rognage, rotations), l'ajout de bruit gaussien et finalement la modification des pixels (luminosité et contraste). Les probabilités d'augmentation par l'un de ces moyens sont élevées afin de permettre un maximum de variation d'une augmentation sur l'autre. Sont données ensuite la proportion du jeu de données initial utilisée pour l'entraînement, le facteur d'augmentation pour les deux groupes de classe et finalement la quantité de données finales disponible pour l'entraînement.

Figure 3: Exemple d'augmentation réalisée sur une image de la classe *multiple maladies*. La dernière image correspond à l'originale uniquement convertie à la résolution $224 * 224$.

2. Parmi les premiers jeux de données augmentés en figurait un augmenté d'un facteur 14. Cela causait un biais, puisque la variance du jeu de données diminuait. En effet, nos algorithmes d'augmentation sont incapables de donner, à partir d'une seule image, quatorze images suffisamment différentes. Nous verrons que cela peut être vrai pour des valeurs inférieurs de facteur d'augmentation (selon l'architecture utilisée ;

3. Finalement, le déséquilibre des classes reste un problème et sera abordé plus en détail ensuite.

2.2 Choix de l'architecture

Pour choisir notre architecture, nous nous sommes donnés un certain nombre d'architectures pré-entraînées sur le jeu de données ImageNet: VGG[3], ResNet[4], GoogLeNet[5], EfficientNet[6].

Nous avons suivi la démarche suivante pour le choix des architectures:

- Tester l'architecture avec un niveau de gel f ;
- Produire la matrice de confusion et visualiser le flot de gradient ;
- Retester avec un niveau de gel f+1.

Les niveaux de gels testés sont: dégel de la dernière couche, dégel des couches denses, dégel d'un tiers des couches, gel des deux premières couches uniquement, dégel total. On dégèle progressivement dans un sens vers les inputs. Les niveaux de gels sont donnés ci-après:

- Gel 1: Dégel de la dernière couche seulement
- Gel 2: Dégel des couches denses
- Gel 3: Dégel d'un tiers des couches
- Gel 4: Gel des deux premières couches uniquement
- Gel 5: Dégel total

Nous donnons dans la table 3 une comparaison des modèles à l'issue de ces tests sur le jeu 6 (qui est de taille moyenne et déséquilibré, voir table 2).

Architecture	Gel 1	Gel 2	Gel 3	Gel 4	Gel 5	#Paramètres (millions)
VGG-16	55	62	32	X	34	138
GoogLeNet	75	53	69	83	79	6
ResNet-50	80	74	73	28	82	25
EfficientNet-b3	66	67	51	X	32	12

Table 3: Tests avec niveau de gel différents. Ce tableau donne, pour chaque niveau de gel, la précision obtenue sur le jeu de données de test. Le nombre de paramètres est également donné.

Nous avons ensuite comparé les architectures[1] gelées au niveau 5 sur la base de leur performance sur un jeu de données plus grand et plus équilibré (Jeu 8 visible en table 2):

ResNet-50 et GoogLeNet affichent une précision de 0.915 et 0.908 respectivement. EfficientNet-b3, lui, affiche 0.754. En revanche, les précisions sont maintenant si élevées qu'elles ne peuvent plus servir de métrique pertinente à elles seules. Nous avons donc produit des matrices de confusion visibles en figure 4. On voit clairement sur la figure 4 que les modèles ont mal appris les *features* caractéristiques de la classe 1 (multiple diseases). Nous avons donc décidé, après avoir vu que rééquilibrer les classes par augmentation ne réussissait pas, d'équilibrer les contributions à l'erreur d'apprentissage de manière inversement proportionnelle à la taille de chaque classe. Ainsi, une erreur sur classe 1 sera beaucoup plus pénalisante qu'une erreur dans une autre classe.

Après avoir effectué ce travail, ResNet-50 affiche 0.842 de précision et EfficientNet-b3 affiche 0.918 de précision. Voici encore une fois les matrices de confusion en la figure 5. Les précisions en phase d'apprentissage sont globalement meilleures avant l'adaptation des coefficients d'erreur. Il n'en demeure pas moins que, malgré sa précision globale élevée, EfficientNet affiche une performance médiocre sur la prédiction de la classe 1 relativement à ResNet qui arrive, près d'une fois sur deux, à détecter correctement les images de cette classe. La meilleure architecture semble bien être ResNet-50.

2.2.1 Modèle à 3 classes

La feuilles de la classe *Multiple maladies* regroupent à la fois les caractéristiques de la classe *Rouille* et *Tavelure*. Ainsi, si une telle feuille est donnée en entrée à un modèle, les neurones s'activant pour *Rouille* (resp. *Tavelure*) s'activeront également pour cette feuille. On s'attend à ce que les images de la classe *multiple maladies* injustement classées le soient dans les deux dernières classes et non pas dans la première.

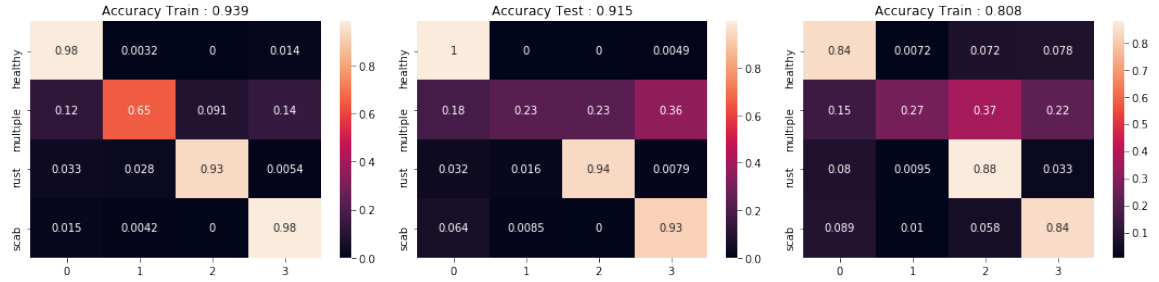
Fort de ce constat, nous avons créé une architecture, fondée sur *ResNet-50*, classifiant cette fois entre trois classes, la classe *multiple maladies* ayant été écartée temporairement. Une fois l'entraînement de ce réseau terminé, nous avons ajouté une nouvelle couche dense permettant de passer de 3 classes à 4 classes et gelé la précédente. De plus, des poids différents ont été attribués à chaque classe : 1 pour les classes *Saine*, *Rouille* et *Tavelure* et 5 pour *Multiple maladies*. L'optimisation Adam a été utilisée.

Puis, lors de l'entraînement, nous avons progressivement dégelé toutes les couches du classifieur en commençant par les dernières. Les résultats sont présentés en partie 3.3.

2.3 Choix des hyperparamètres et outils d'analyse

Nous avons optimisé les hyperparamètres suivant pour nos architectures: le nombre d'itérations (*epochs*) et le taux d'apprentissage (*learning rate*). Nous avons également utilisé deux outils pour évaluer la correction de nos modèles: courbes d'apprentissage et de validation, et flot de gradient.

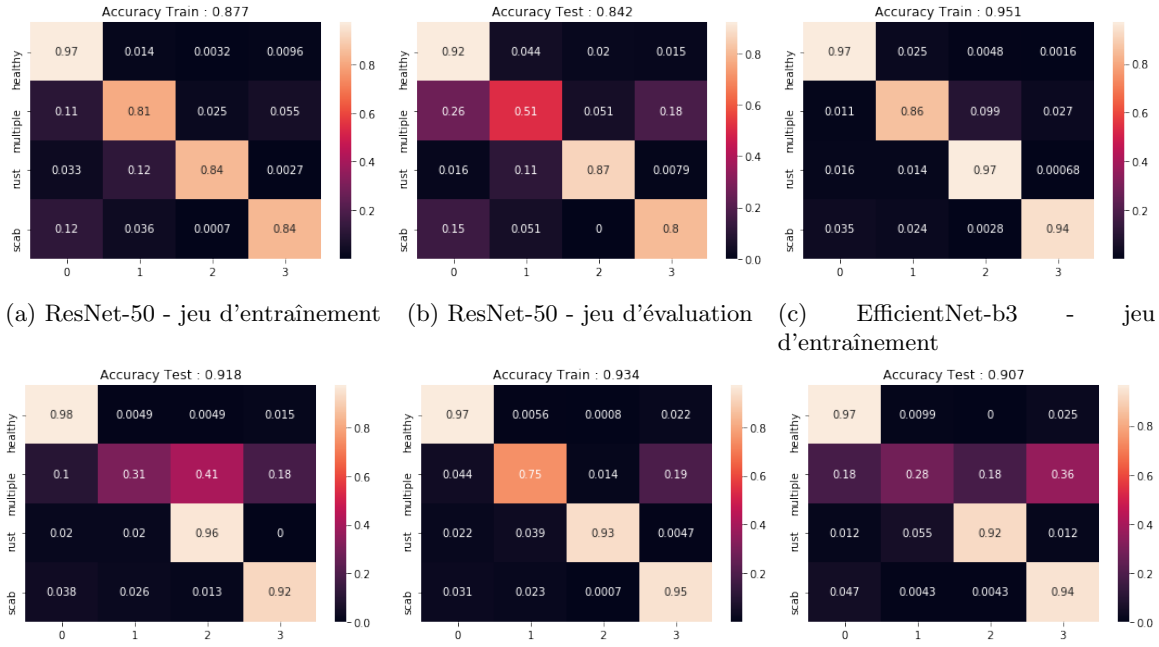
[1]Nous ne montrons pas les résultats pour l'architecture VGG-16 car non satisfaisants à cause de la trop faible précision relativement au nombre de paramètres.



(a) ResNet-50 - jeu d'entraînement (b) ResNet-50 - jeu d'évaluation (c) EfficientNet-b3 - jeu d'entraînement

(d) EfficientNet-b3 - jeu d'évaluation (e) GoogLeNet - jeu d'entraînement (f) GoogLeNet - jeu d'évaluation

Figure 4: Matrices de confusion avant adaptation des coefficients d'erreur. Nous remarquons bien que la classe 1 est très mal apprise par les deux modèles. Le choix est presque aléatoire. Il est aussi utile de noter que ResNet et GoogLeNet font du sur-apprentissage sur la classe 1 mais ne généralisent pas correctement sur le jeu d'évaluation.



(a) ResNet-50 - jeu d'entraînement (b) ResNet-50 - jeu d'évaluation (c) EfficientNet-b3 - jeu d'entraînement

(d) EfficientNet-b3 - jeu d'évaluation (e) GoogLeNet - jeu d'entraînement (f) GoogLeNet - jeu d'évaluation

Figure 5: Matrices de confusion après adaptation des coefficients d'erreur. Les précisions en phase d'apprentissage sont largement meilleures avant l'adaptation des coefficients d'erreur. Il n'en demeure pas moins que, malgré sa précision globale élevée, EfficientNet affiche une performance médiocre sur la prédiction de la classe 1 relativement à ResNet qui arrive, près d'une fois sur deux, à détecter correctement les images de cette classe.

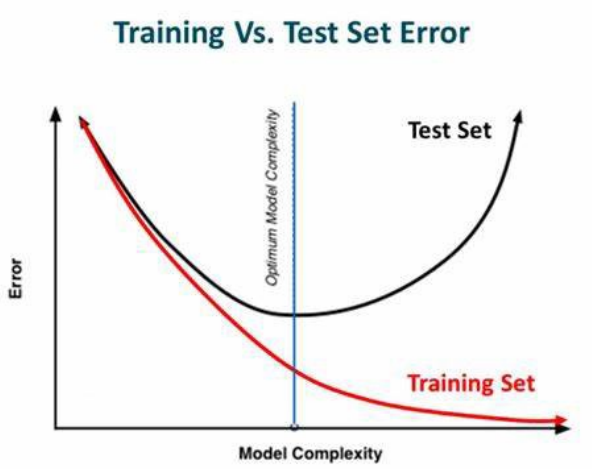


Figure 6: Figure montrant la technique Early Stopping sur les courbes d'apprentissage. Credits: `https://medium.com/ai%C2%B3-theory-practice-business/dropout-early-stopping-188a23beb2f9`

2.3.1 Optimisation du nombre d'*epochs*

Nous avons utilisé la technique du Early Stopping. Cette technique consiste à garder une sauvegarde du modèle disposant une perte de validation minimale sur les *epochs* d'entraînement. De plus, cette technique fait que le processus d'entraînement s'arrête, avec un paramètre de patience, dès que la perte de validation n'évolue plus. Ainsi, on arrête l'apprentissage à un niveau au delà duquel on ferait de l'*overfitting*. La technique est illustrée figure 6.

2.3.2 Optimisation du learning rate

Nous avons utilisé le module *pytorch-lr-finder* qui implémente le papier suivant *"Cyclical Learning Rates for Training Neural Networks"* de Leslie Smith, [7]. La technique utilisée consiste en le parcours (exponentiel ou linéaire) de différents taux d'apprentissage, du plus petit au plus grand, et de suivre l'évolution de la fonction de perte sur quelques itérations (de l'ordre de 5 *epochs* généralement). Nous disposons ensuite, grâce au module cité plus haut, une courbe similaire à celle de la figure 7. Sur cette courbe, nous choisissons autant que possible un point précédent le minimum global et ayant la plus grande pente possible. Avec ce taux d'apprentissage, on est sûr de converger rapidement vers le minimum de la perte.

2.3.3 Outils d'analyse

Comme cité précédemment, nous avons utilisé deux outils pour suivre et évaluer la correction de nos modèles. Le premier outil est le suivi des courbes d'apprentissages et des fonctions de perte (voir figure 8) qui nous permettait d'évaluer le degré d'*overfitting* ou encore la qualité de l'*early stopping*. Le deuxième est la visualisation du flot de gradient (voir figure 9). Ce dernier était particulièrement utile pour vérifier s'il y avait une explosion/disparation de gradients.

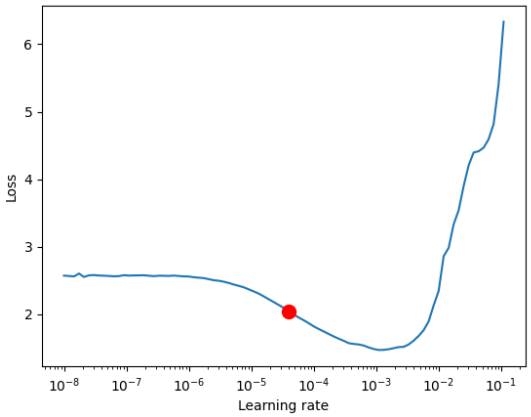


Figure 7: Figure montrant le graphe de la perte en fonction du taux d'apprentissage. Les meilleurs taux d'apprentissage se trouvent dans la région autour du point en rouge. Credits: `https://pytorch-lightning.readthedocs.io/en/latest/lr_finder.html`

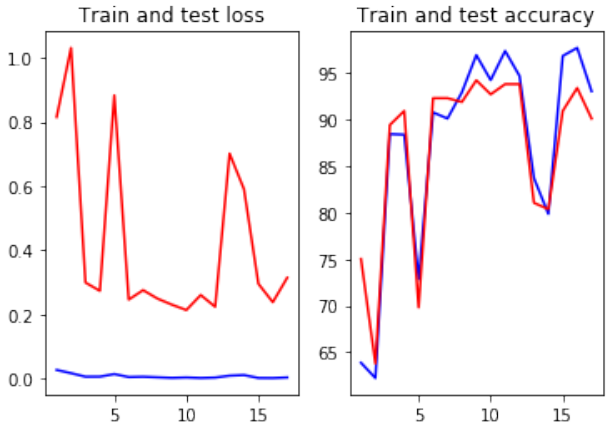


Figure 8: Figure montrant les courbes de perte (à gauche) et de précision (à droite) lors de l'apprentissage en fonction du nombres d'itérations (epochs). En rouge les courbes de validation, en bleu les courbes d'entraînement.

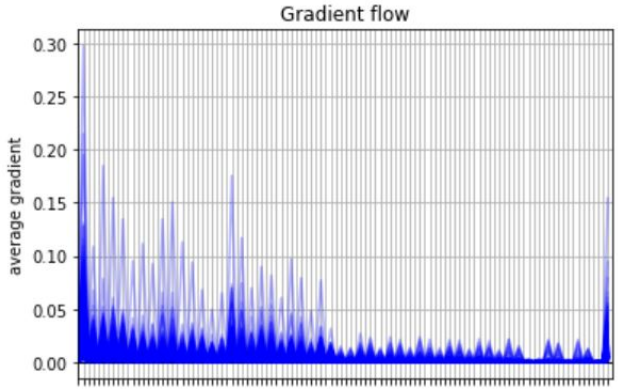


Figure 9: Figure montrant le flot de gradient à travers les différentes couches de l'architecture. Plus le gradient est élevé, plus la couche concernée apprend plus rapidement.

Architecture	Précision locale	Précision Kaggle
GoogLeNet (erreur équilibrée)	0.91	0.91
GoogLeNet	0.91	0.93
ResNet-50 (erreur équilibrée)	0.84	0.83
ResNet-50	0.92	0.93
EfficientNet-b3 (erreur équilibrée)	0.92	0.83
EfficientNet-b3	0.75	0.88

Table 4: Précisions des modèles en local et en ligne

3 Résultats

3.1 Organisation des résultats

Du fait du faible nombre de représentant de la classe "multiple maladies", il est possible d'obtenir une bonne précision tout en ignorant cette classe. Le choix quant à l'architecture se fait donc en fonction des objectifs. En effet, comme le suggère le jeu de données, les feuilles présentant de multiples maladies sont probablement en proportion faible dans la nature.

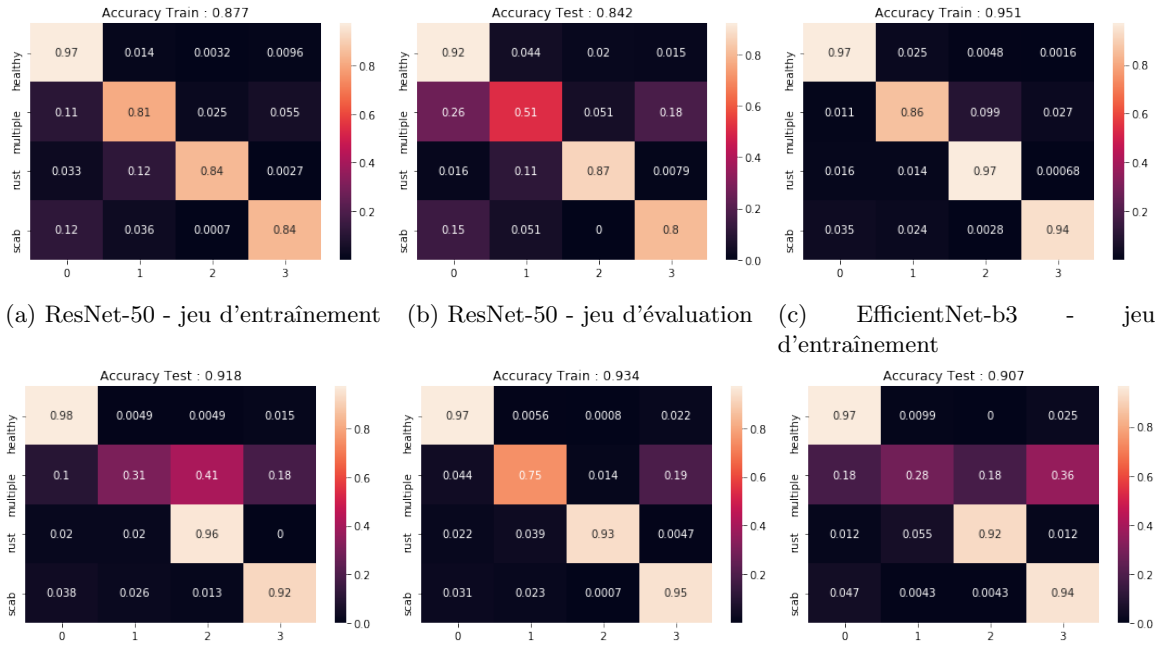
Ainsi, on peut considérer que si on reconnaît déjà précisément les classes Tavelure, Rouille et Saine, alors on traite la majorité des cas et cela suffit.

On peut à l'inverse considérer que c'est justement le cas de multiple maladies qui est pertinent ici et on voudrait alors le reconnaître avec précision. De cet objectif dépendra l'architecture. Pour ces raisons, nous vous proposons deux architectures : une équilibrée entre les quatre classes et une seconde qui cherche à maximiser sa précision.

3.2 Architectures 4 classes

Les résultats pour les architectures avec 4 classes en sortie sont donnés en figure 10. Sur la table 4, les précisions en local et sur le jeu de validation de la compétition Kaggle sont donnés. Il est utile de remarquer que:

- Les précisions sont très proches ce qui suggère que nos jeu de données sont de bonne qualité
- On ne peut pas savoir laquelle des deux précisions est une meilleure métrique pour évaluer les modèles. Il n'en demeure pas moins que, en local, nous disposons des matrices de confusion qui nous en disent plus que la seule précision Kaggle.



(a) ResNet-50 - jeu d'entraînement (b) ResNet-50 - jeu d'évaluation (c) EfficientNet-b3 - jeu d'entraînement

(d) EfficientNet-b3 - jeu d'évaluation (e) GoogLeNet - jeu d'entraînement (f) GoogLeNet - jeu d'évaluation

Figure 10: Résultats architectures avec 4 classes.

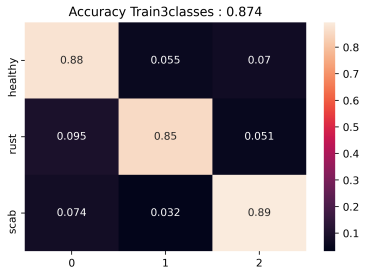
- Nous n'avons fait aucun *fine-tuning* car ce n'était pas le but du projet. Des précisions de l'ordre de 93 sont donc très encourageantes.

3.3 Architecture ResNet-50 - 3 classes

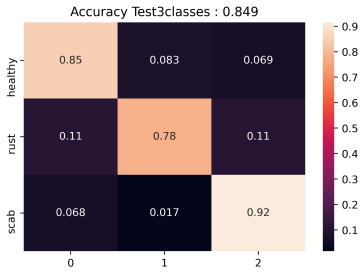
Pour ces résultats, nous avons utilisé les jeux de données intitulés 3 classes puis le jeu de données numéro 8. Les résultats que nous avons obtenus sont donnés en figure 11.

Ils ne sont pas aussi concluants que dans le cas d'une classification sur quatre classes. La précision de ce modèle donne 82% sur le jeu d'évaluation de la compétition Kaggle.

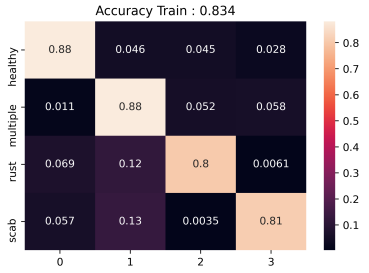
Un sur-apprentissage sur la classe *multiple maladies* est visible. On pourrait probablement améliorer le résultat en utilisant un jeu de données pour lequel l'augmentation sur la classe multiple maladies est moins conséquente.



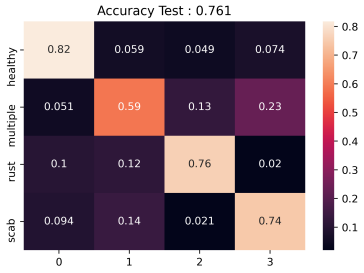
(a) Jeu d'entraînement



(b) Jeu d'évaluation



(c) Jeu d'entraînement - phase 4 classes



(d) Jeu d'évaluation - phase 4 classes

Figure 11: Résultats sur *ResNet-50* avec toutes les couches gelées et pour 35 epochs. Pour la matrice de confusion, le vraies classes sont notées en ordonnées, la somme sur une ligne des probabilités fait 1.

4 Conclusion et extension

L'un des enjeux de ce projet était le jeu de données et son déséquilibre entre les classes. Nous avons donc procédé à une augmentation des données.

Plusieurs architectures ont été explorées, en gelant à différent degrés leurs couches. Parmi ces architectures, ResNet-50 et GoogLeNet donnent de bons résultats, autant sur les jeu de données locaux que sur le jeu de données de la compétition. Avec ou sans adaptation des erreurs, les résultats sont différents à apprécier: les précisions sont meilleures sans adaptation des erreurs, mais les confusions sur les classes sous-représentées sont beaucoup moins fréquentes, le modèle est ainsi plus généralisable. Durant nos recherches, nous avons été amenés à implémenter des méthodes classiques du Machine Learning, notamment l'early stopping et l'ajout de poids dans la fonction de perte pour compenser le déséquilibre.

Des améliorations sont possibles. Nous avons principalement procédé jusqu'à présent à un large défrichage des architectures et jeu de données donnant des résultats corrects pour ce problème. Une deuxième phase consisterait à sélectionner le couple (jeu de données, architecture) qui présente le plus de potentiel et à l'optimiser le plus finement possible. Cela pourrait notamment passer par la création d'un jeu de données sur-mesure pour l'architecture, avec notamment les images problématiques enlevées. En effet, un participant à la compétition Kaggle a par exemple relevé la présence d'une même image deux fois, classées dans deux classes distinctes, dans le jeu de données. Nous avons également essayer d'utiliser des modèles pré-entraînés sur des tâches de reconnaissance de type de plantes, mais certaines difficultés techniques nous ont empêché de les exploité correctement. De plus, ImageNet est suffisamment large et varié pour que les modèles qui sont entraînés sur sa base de données soient transférables, à condition de dégeler correctement.

Références

[1] Ranjita Thapa, Noah Snavely, Serge Belongie, and Awais Khan. The plant pathology 2020 challenge dataset to classify foliar disease of apples, 2020.

[2] Kaggle - plant pathology 2020 - Link : https://www.kaggle.com/c/plant-pathology-2020-fgvc7/overview.

[3] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition, 2014.

[4] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015.

[5] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions, 2014.

[6] Mingxing Tan and Quoc V. Le. Efficientnet: Rethinking model scaling for convolutional neural networks, 2019.

[7] Leslie N. Smith. Cyclical learning rates for training neural networks, 2015.