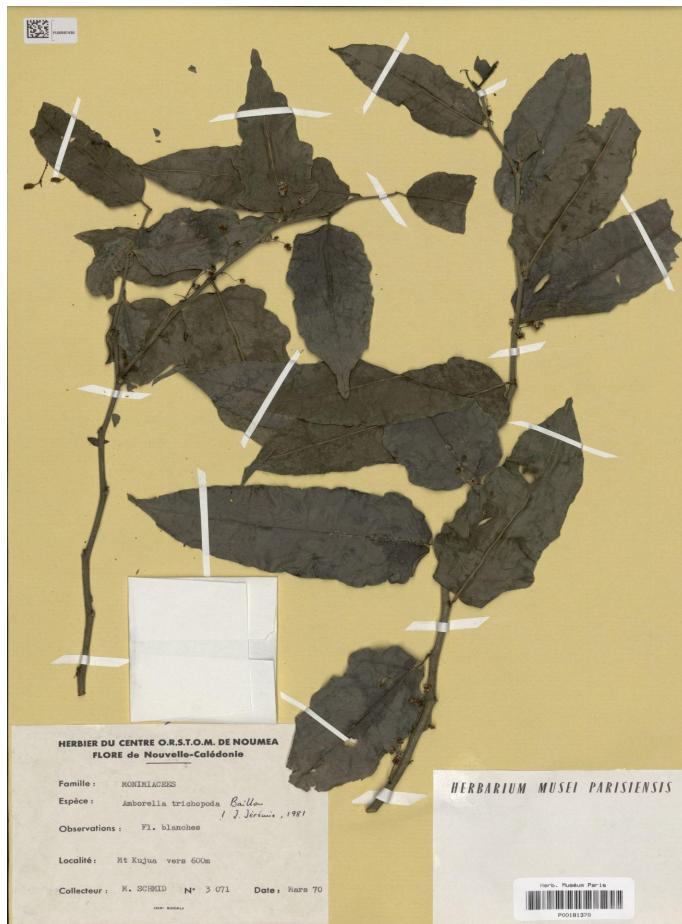


Identification des Caractères des Plantes



Julien **DELAGES** - Loic **ESNAULT**, Fedi **LAHMAR**,
Natan **LE BOURDONNEC**, Dorian **ROUX**



23 Décembre 2022

Cursus Ingénieur – Option IA - Groupe B – Promotion 2023

Sommaire

Notre rapport est décomposé en plusieurs parties comme visible ci-dessous permettant une meilleure compréhension de son contenu.

I. Introduction	2
II. Méthodes	3
a. Jeu de Données	3
b. Traitement des Images	5
c. Modèle de Deep Learning	6
d. Métriques	8
e. Outils	8
f. Répartition des tâches	9
III. Résultats	10
a. Traitement des Images, Semi-Automatisé	10
b. Traitement des Images avec Automatisé	13
c. Apprentissage	18
IV. Conclusion	22
a. Traitement de l'Image	22
b. Apprentissage	22
c. Difficultés Rencontrées	23
V. Perspectives	24

I. Introduction

L'identification des plantes est une tâche cruciale dans de nombreux domaines, tels que l'agriculture, la foresterie, l'horticulture et la botanique. Elle peut être utile pour la gestion des ressources naturelles, la protection de la biodiversité, la recherche scientifique, la médecine traditionnelle et la découverte de nouvelles espèces. Cependant, cette tâche peut s'avérer complexe et nécessiter l'expertise de spécialistes, en particulier lorsque les plantes sont jeunes ou en milieu sauvage.

Pour résoudre ces problèmes de reconnaissance des plantes, des solutions à l'aide d'Intelligence Artificielle peuvent être envisagées. Ces dernières offrent de nombreuses possibilités pour résoudre ces problèmes de reconnaissance des plantes. En utilisant des réseaux de neurones et des techniques de traitement d'images, il est possible de développer des modèles capables de reconnaître et de classer automatiquement les plantes à partir de leurs caractères morphologiques.

Pour cela, il est important d'être méticuleux sur le traitement d'images, qui est une discipline importante du deep learning permettant de traiter et d'analyser des données visuelles de manière automatique. Cela permet d'extraire des caractéristiques, des patterns et des informations utiles à partir d'images. Dans le domaine de l'identification des plantes, le traitement d'images peut être utilisé pour analyser les feuilles et les tiges des plantes, afin de déterminer leur espèce, leur famille, leur genre et leur variété.

Dans ce projet, nous proposons d'utiliser un dataset d'herbiers pour entraîner et évaluer un modèle de deep learning capable d'identifier les caractères des plantes. Dans ce rapport, nous présentons les différentes étapes de préparation et de traitement des données, ainsi que les choix de modèle et d'hyper paramètres. Nous utilisons des techniques de preprocessing, de segmentation et de feature extraction pour extraire des caractéristiques pertinentes des images. Nous entraînons ensuite un réseau de neurones convolutionnel sur ces caractéristiques, en utilisant une approche d'apprentissage supervisé. Nous mesurons les performances du modèle sur différentes tâches de classification et nous discutons de ses limites et de ses perspectives d'application.

II. Méthodes

Dans cette partie nous allons présenter les différentes méthodes utilisées dans notre projet. Nous ferons une description du jeu de données ainsi que des méthodes de traitement de l'image que nous lui avons appliquée. Puis, nous allons expliquer les méthodes liées au modèle de Deep Learning (DL) ainsi qu'une définition des métriques utilisées à son évaluation. Nous allons aussi définir les différents outils nécessaires à la réalisation de notre projet et la répartition des tâches.

a. Jeu de Données

Dans la réalisation de notre projet, nous disposons d'un jeu de données contenant 330 images d'herbier réparties en 11 groupes de plantes différentes. Ce dataset comprend une partie **entraînement** avec 20 images par groupe, pour un total de 220 images et une partie **test** avec 10 images par groupe, pour un total de 110 images. Nous pouvons observer la figure suivante qui représente l'exemple d'un herbier dont l'espèce est la monimiaceae:

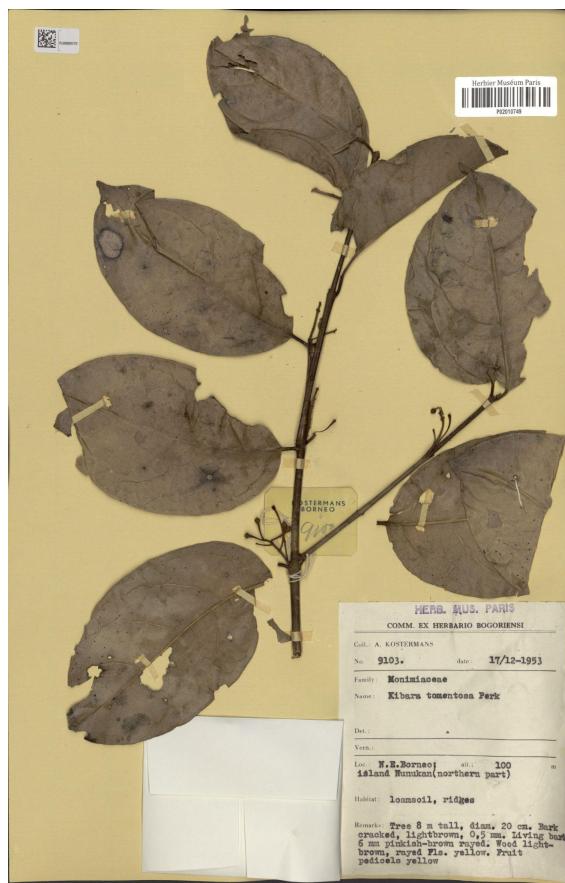


Figure 1. Exemple d'un herbier de l'espèce monimiaceae.

Nous pouvons observer sur cet herbier différentes informations avec notamment les éléments de la plante comme la tige, les feuilles, les bordures mais aussi des éléments informatifs comme les étiquettes, les timbres, les annotations, etc.

Comme dit précédemment, notre jeu de données est composé de 11 groupes de plantes différentes avec pour chaque groupe des caractéristiques différentes. Dans notre étude, nous en avons considéré uniquement 4 étant le type de bord du limbe, la disposition des feuilles, le type de feuille et le type végétal. Nous pouvons observer la table suivante qui représente les différents caractères descriptifs des 11 groupes:

Nom	Bord	Phyllotaxie	Type feuille	Ligneux
Convolvulaceae	lisse	alterné	simple	non
Monimiaceae	lisse	opposé	simple	oui
<i>Amborella</i>	lisse	alterné	simple	oui
<i>Castarea</i>	denté	alterné	simple	oui
<i>Desmodium</i>	lisse	alterné	composée	non
<i>Eugenia</i>	lisse	opposé	simple	oui
<i>Laurus</i>	lisse	opposé	simple	oui
<i>Litsea</i>	lisse	alterné	simple	oui
<i>Magnolia</i>	lisse	alterné	simple	oui
<i>Rubus</i>	denté	alterné	composée	oui
<i>Ulmus</i>	denté	alterné	simple	oui

Table 1. Taxons et caractères descriptifs.

D'après la table ci-dessus, nous pouvons remarquer un manque d'équilibrage entre les caractères, ce qui peut avoir un impact sur les performances générales du modèle. Afin d'avoir une meilleure compréhension, nous pouvons observer la figure suivante qui montre la représentation de chaque classe des caractères descriptifs de la plante.

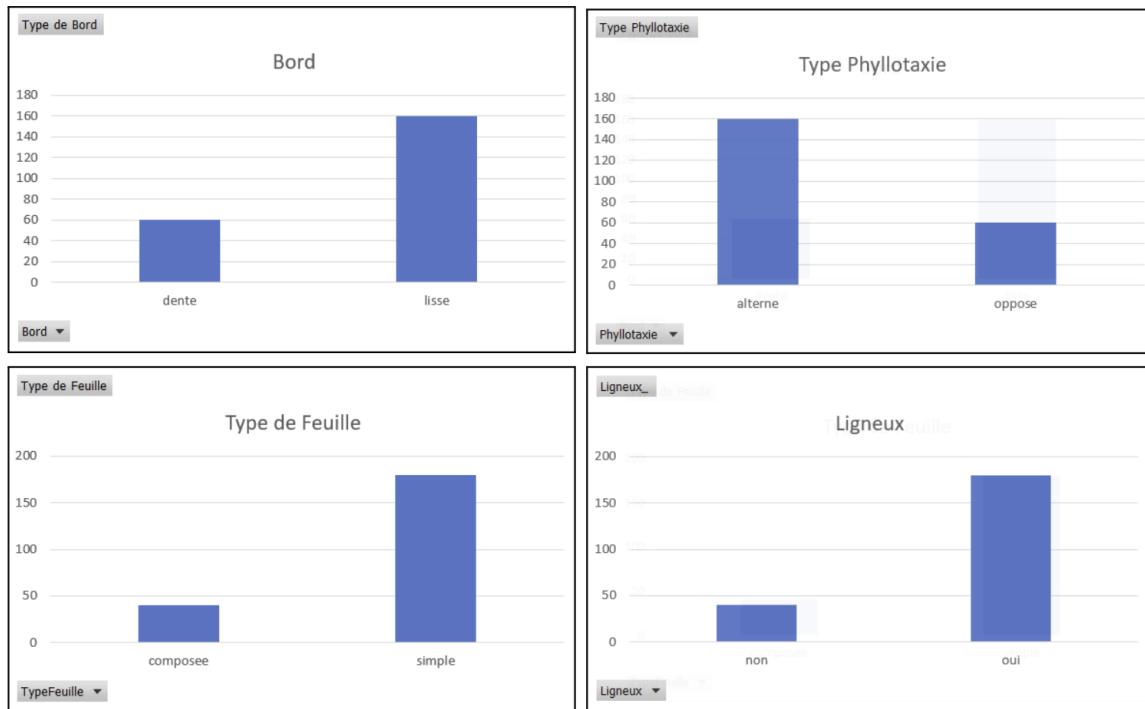


Figure 2. Présence des classes de chaque caractère descriptif étudié

Comme attendu, certaines classes sont sous-représentées tandis que d'autres sont sur-représentées. Ainsi, le modèle risque de ne pas être exposé à suffisamment d'exemples de certaines classes, et de ne pas être capable de les prédire de manière adéquate.

b. Traitement des Images

Dans la sous-partie précédente, nous avons expliqué le jeu de données à notre disposition, et, nous avons pu remarquer que ces images ont énormément d'informations dont certaines inutiles notamment les étiquettes, timbres, etc. Aussi, une tâche nécessaire avant d'entraîner notre modèle est la préparation des images.

Ce traitement permet une meilleure homogénéisation et normalisation des images ainsi que le retrait de bruits potentiel ce qui a pour objectif d'améliorer l'apprentissage du modèle. Dans notre cas, la tâche la plus importante du traitement de l'image est de retirer les code-barres, les étiquettes ainsi que les papiers rectangulaires qui ajoutent des informations et un bruit non nécessaire dans notre modélisation.

Pour la réalisation du traitement d'images, nous avons essayé deux méthodes différentes dont nous expliquerons les performances et résultats dans les parties suivantes. La première méthode consiste à une semi-automatisation et peut être réalisé avec les étapes suivantes:

1. Calcul de l'histogramme de l'image avec le repérage du pic associé au fond de l'image et celui des objets à retirer.
 2. Sélection d'une valeur de niveaux permettant de séparer ces pics.
 3. Conversion de l'image en niveaux de gris.
 4. Binarisation de l'image en niveaux de gris via un threshold avec la valeur de l'étape (2) permettant la séparation des pics de l'histogramme.
 5. Dilatation de la binarisation pour enlever le bruit et avoir des zones plus lisses.
 6. Remplacement des pixels de la dilatation par la couleur moyenne du fond de l'image.
- Cette méthode permet de remplacer les objets à enlever par la couleur moyenne du fond.

Cette méthode à des avantages et inconvénients dont nous discuterons plus tard. Aussi, nous avons décidé de travailler sur une seconde méthode qui consiste à une automatisation complète et pouvant être réalisé avec les étapes suivantes:

1. Conversion de l'image en niveau de gris.
2. Application d'un filtre Gaussien sur l'image en niveau de gris.
3. Binarisation de l'image filtrée à l'aide d'un threshold Binaire et Otsu.
4. Colorisation de l'image binarisée
5. Rognage de l'image colorisée
6. Redimensionnement de l'image rogner
7. Remplacement des pixels « noires » par les nuances de couleurs majoritaire étant le fond.

c. Modèle de Deep Learning

Dans la réalisation de notre projet, comme nous n'avions que très peu d'images d'entraînement pour chaque groupe, nous avons décidé de construire un modèle de classification afin de l'utiliser sur chacun des caractères descriptifs des plantes. Nous pouvons observer la figure suivante qui représente l'architecture de notre modèle de classification.

Layer (type)	Output Shape	Param #
<hr/>		
conv2d_30 (Conv2D)	(None, 510, 510, 32)	896
max_pooling2d_30 (MaxPooling2D)	(None, 255, 255, 32)	0
conv2d_31 (Conv2D)	(None, 253, 253, 64)	18496
max_pooling2d_31 (MaxPooling2D)	(None, 126, 126, 64)	0
conv2d_32 (Conv2D)	(None, 124, 124, 128)	73856
max_pooling2d_32 (MaxPooling2D)	(None, 62, 62, 128)	0
conv2d_33 (Conv2D)	(None, 60, 60, 256)	295168
max_pooling2d_33 (MaxPooling2D)	(None, 30, 30, 256)	0
conv2d_34 (Conv2D)	(None, 28, 28, 512)	1180160
max_pooling2d_34 (MaxPooling2D)	(None, 14, 14, 512)	0
flatten_6 (Flatten)	(None, 100352)	0
dropout_12 (Dropout)	(None, 100352)	0
dense_18 (Dense)	(None, 64)	6422592
dropout_13 (Dropout)	(None, 64)	0
dense_19 (Dense)	(None, 32)	2080
dense_20 (Dense)	(None, 1)	33

Figure 3. L'architecture du modèle de classification avec les différentes couches.

Ce modèle est un Convolutional Neural Network (CNN) composé de différentes couches de convolutions.

Les couches Conv2D et l'association MaxPooling2D sont répétées à 5 reprises. Pour chacune des couches de convolutions, nous avons utilisé la fonction d'activation **ReLU** dont la formule est la suivante:

$$f(x) = \max(0, x)$$

En d'autres mots, la fonction renvoie comme valeur 0 lorsque x est inférieur ou égal à 0 et la fonction renvoie comme valeur x lorsque x est supérieur à 0. Nous pouvons observer la figure suivante qui représente la courbe de cette fonction d'activation

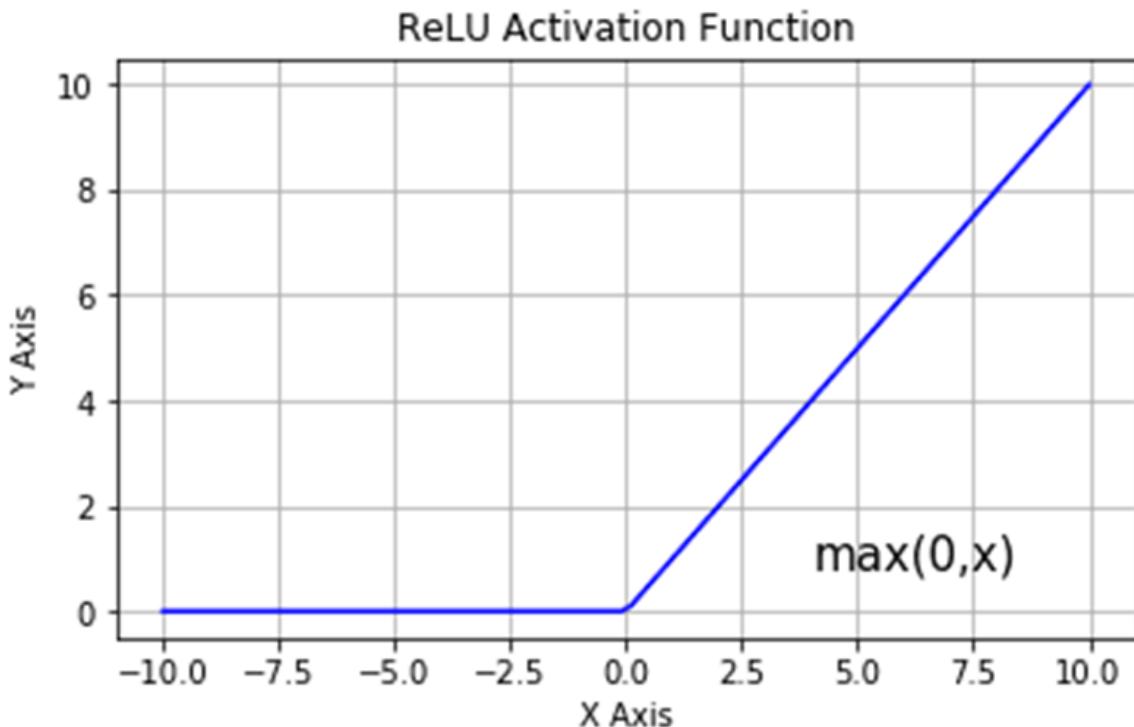


Figure 4. La fonction d'activation ReLU

Les couches finales du réseau sont composées d'un Flatten, Dropout et Dense afin de manipuler la taille de l'output ainsi qu'atténuer le sur-apprentissage.

Dans notre modèle, nous avons utilisé comme fonction de perte la **sparse categorical cross entropy** qui permet de calculer la différence entre les prédictions d'un modèle et les valeurs réelles, dans les cas où les valeurs cibles représentent des classes. La formule de cette fonction est la suivante:

$$Loss = - \sum_{i=1}^{\text{output size}} y_i \cdot \log(\hat{y}_i)$$

Cette fonction de perte permet de mesurer la précision du modèle et peut être utilisée pour minimiser la différence entre les valeurs prédites et les valeurs réelles, ce qui permet d'entraîner le modèle pour qu'il produise des prédictions plus précises.

d. Métriques

Les métriques sont des indicateurs ou des mesures utilisés pour évaluer la performance ou la qualité d'un modèle de deep learning. Elles permettent de quantifier de manière objective et comparative les résultats obtenus par le modèle sur des données de test ou de validation, et de les comparer à des critères de référence ou à des objectifs prédéfinis. Dans le calcul des performances de notre modèle, nous avons utilisées les métriques suivantes:

- L'**accuracy** étant le pourcentage de prédictions correctes sur toutes les prédictions faites lors de l'évaluation d'un modèle.
- La **précision** étant le pourcentage de prédictions positives correctes sur tous les cas classés comme positifs. En d'autres termes, le rapport des vrais positifs sur la somme des faux positifs et des vrais négatifs.
- La **sensibilité** (recall) mesurant la capacité du modèle à détecter tous les éléments positifs d'un dataset. Elle est souvent utilisée pour les tâches de détection ou de diagnostic, et s'exprime en pourcentage.
- Le **score F1** mesurant l'équilibre entre la précision et la sensibilité du modèle. Il est souvent utilisé lorsque les classes du dataset sont déséquilibrées ou lorsque les erreurs de type I et de type II ont des conséquences différentes ou inégales.
- La **matrice de confusion** (confusion matrix) étant un tableau qui résume les prédictions du modèle par rapport aux valeurs réelles des données. Elle est souvent utilisée pour les tâches de classification multi-classe, et permet de visualiser les erreurs de prédiction de chaque classe.

e. Outils

Dans la réalisation de notre projet, nous avons utilisé divers outils aussi bien techniques que collaboratifs. Parmi lesquels:

- Python en tant que langage de programmation, pour la construction de l'ensemble de notre projet. À travers ce langage, nous avons utilisé un grand nombre de libraires comme:
 - OpenCV en tant que bibliothèque de traitement d'images et de vision par ordinateur en libre accès. Elle permet de charger, afficher et manipuler des images, de détecter des formes et des caractéristiques, de faire du suivi et de la reconnaissance d'objets, de la segmentation.
 - Seaborn en tant que bibliothèque de visualisation de données en Python basée sur Matplotlib. Elle est particulièrement utile pour l'exploration et la visualisation de données statistiques.
 - Scikit-Learn en tant que bibliothèque de machine learning en Python. Elle est basée sur des structures de données de NumPy et de SciPy et est facilement intégrable avec d'autres bibliothèques Python.
 - Tensorflow en tant que framework de machine learning. Il permet de définir, entraîner et évaluer des modèles de deep learning. Il offre une grande flexibilité et une haute performance sur différentes plateformes et architectures matérielles.

- Google Colab en tant que plateforme pour développer, entraîner et évaluer des modèles de réseaux de neurones sur des datasets volumineux ou complexes, sans avoir à se soucier de la configuration ou de l'optimisation de l'environnement de travail en se connectant à des GPUs ou TPUs (Tensor Processing Units) pour accélérer le calcul et améliorer les performances.
- Github pour l'hébergement et le partage de notre code source.
- Google Drive pour le stockage et le partage de fichiers en ligne.
- Microsoft Teams pour la communication et la collaboration entre membres du groupe.

f. Répartition des tâches

Dans la réalisation du projet, nous avons tous effectué certaines tâches ainsi que certaines tâches identiques effectuées par plusieurs membres. Voici une liste des tâches principales effectués par chaque membre du groupe:

Travail effectué par Dorian:

- Gestion de l'entièreté du projet et de son bon fonctionnement sur Github.
- Réalisation de la seconde méthode de traitement des images avec performance moyenne et automatisation.
- Construction des datasets Train, Test et Validation.
- Construction du *premier modèle* avant l'ajout de l'option Data Augmentation.

Travail effectué par Julien:

- Réalisation de la première méthode de traitement des images avec performance élevée mais sans automatisation.
- Amélioration du *modèle* ainsi que l'ajout de l'option Data Augmentation.
- Construction des différentes métriques (Learning Rate, Loss Curve, Confusion Matrix).

Travail effectué par Loïc:

- Réalisation de méthode de traitement d'image par seuillage binaire qui donnait des performances peu satisfaisantes étant donné la variété de couleurs des fonds.
- Réalisation de méthode de data augmentation qui n'a pas servi.

Travail effectué par Fedi:

- Construction des différentes métriques (Learning Rate, Loss Curve, Confusion Matrix).

Travail effectué par Natan:

-

III. Résultats

Nous venons d'introduire les méthodes de notre projet, nous allons à présent expliquer les les différents résultats obtenus à travers le traitement de l'image ainsi que les métriques liées à l'entraînement et aux prédictions de notre modèle de DL.

a. Traitement des Images, Semi-Automatisé

Dans cette sous-partie, nous allons étudier le traitement de l'image semi-automatisé avec la première méthode décrite précédemment sur l'**amborella081** provenant du jeu de données **Train** et que nous pouvons observer ci-dessous:



Figure 5. Image amborella 081 originale

La première étape consiste à calculer l'histogramme de l'image et de repérer le pic associé au fond de l'image et celui des objets à retirer. Nous pouvons observer la figure suivante qui représente l'histogramme de l'image originale:

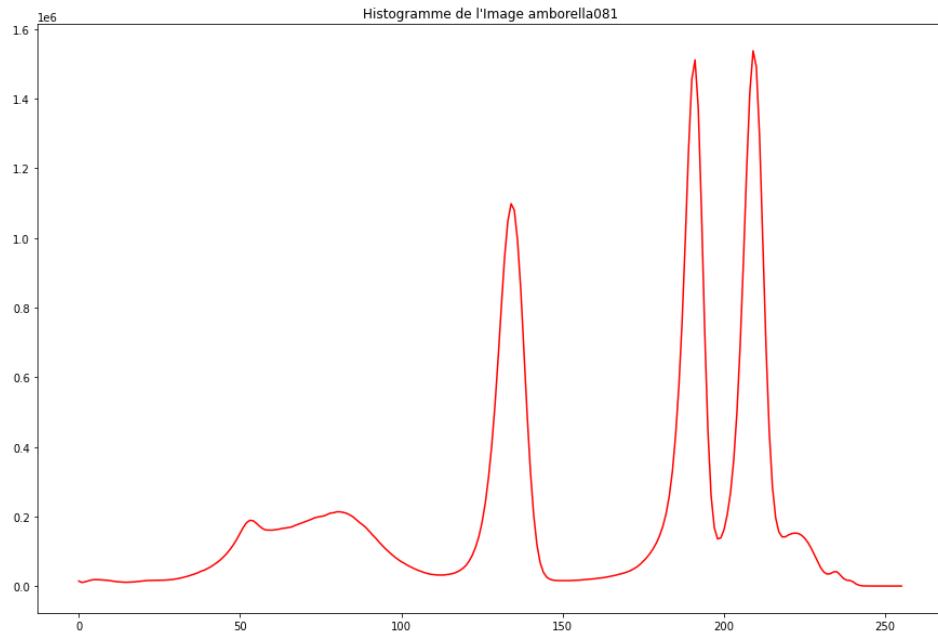


Figure 6. Histogramme de l'image amborella081

Sur la Figure 6, on peut observer deux extreums à approximativement 180 et 220. Ces extreums correspondent respectivement au fond de l'image et aux objets à retirer.

L'étape 2 consiste à trouver une valeur de seuil. Nous pouvons observer à 200, que l'on peut distinguer ces deux courbes, il s'agira ici de notre threshold (seuil) pour l'étape 4. À l'étape 3, nous allons convertir l'image originale en niveaux de gris (Figure 7).

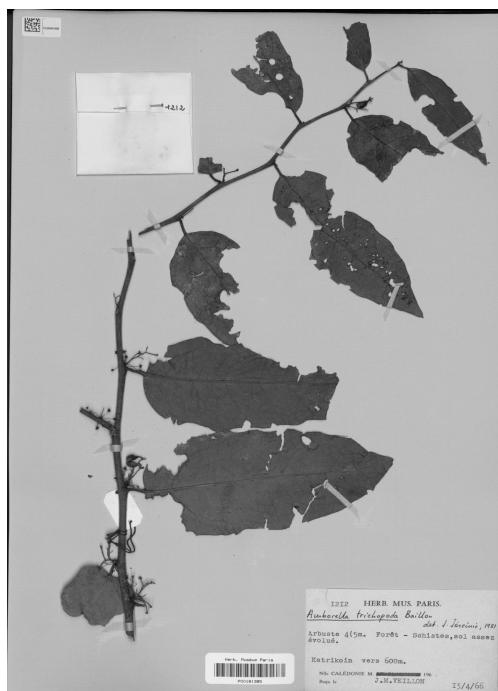


Figure 7. Image amborella081 convertie en niveau de gris

Ensuite nous appliquons la binarisation (threshold) séparant les objets du reste de l'image grâce à l'étape 2 (Figure 8).

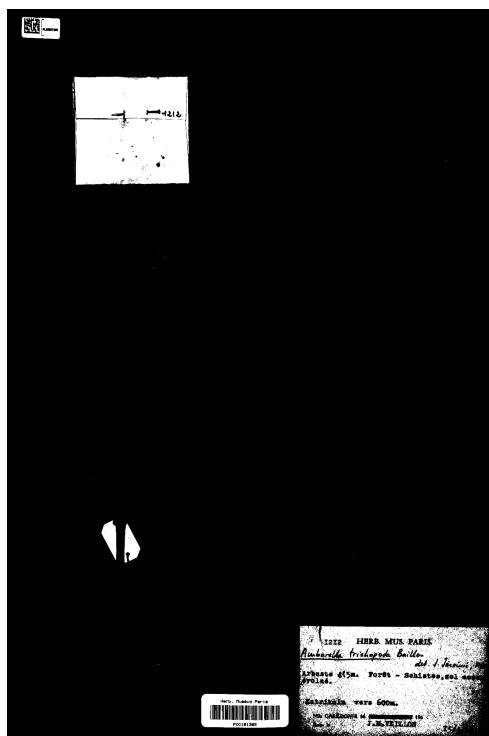


Figure 8. Image amborella081 binarisée

Une fois cette cinquième étape effectuée nous allons procéder à l'application d'un algorithme de dilatation afin d'homogénéiser ou lisser (remplir les trous) les objets à enlever (Figure 9).



Figure 9. Image amborella081 binarisée et dilatée

Enfin, nous remplaçons les pixels des objets à retirer par la couleur moyenne du fond de l'image (dans notre cas une sorte de beige). Cela nous permettra d'obtenir l'image traitée finale (Figure 10).



Figure 10. Image amborella 081 après traitement complet

b. Traitement des Images avec Automatisé

Dans cette sous-partie, nous allons étudier le traitement de l'image **amborella075** provenant du jeu de données Test et que nous pouvons observer ci-dessous:



Figure 11. Image amborella075 originale

Les images du jeu de données sont des images en 3-dimensions. Afin de faciliter le traitement de ces images, nous les avons converties en images 2D. Pour ce faire, nous avons utilisé la formule suivante appliquée à chaque pixels de l'image:

$$Pixel_{X,Y} = \frac{1}{3} \times \sum(Pixel_{X,Y-\text{Rouge}} + Pixel_{X,Y-\text{Vert}} + Pixel_{X,Y-\text{Bleu}}), \text{ avec}$$

X , le pixel correspondant à la hauteur,
 Y , le pixel correspondant à la largeur

Par exemple, imaginons une image en 3-dimension de taille (512,512) et que nous voulons la transformer en 2-dimension, alors, si nous définissons le premier pixel de l'image comme étant [255, 255, 255] alors celui-ci devient 255.

Ainsi, nous pouvons appliquer cette formule sur l'ensemble des pixels de l'images et obtenir la figure suivante:



Figure 12. Image amborella075 convertie en niveau de gris

Nous pouvons remarquer que l'image reste identique dans ses attributs mais que les nuances sont désormais uniquement entre blanc (pixel = 255) et noir (pixel = 0).

Une fois l'image convertie en 2-dimensions, nous pouvons lui appliquer un threshold binaire et otsu.

Le threshold binaire permet d'attribuer la valeur 0 à tous les pixels de l'image étant en dessous d'un threshold prédéfini et d'attribuer la valeur 1 ou 255 à tous les pixels étant au-dessus de ce même threshold.

D'un autre côté, la méthode d'Otsu a pour objectif d'effectuer un seuillage automatique à partir de la forme de l'histogramme de l'image. La supposition est que l'image à binariser ne contient que deux classes étant les objets et l'arrière-plan.

L'algorithme d'Otsu consiste à réaliser de manière itératif le calcul du seuil optimal T qui sépare les deux classes afin que la variance intra-classe soit minimale et que la variance inter-classe soit maximale.

$$\sigma_w^2 = \omega_{1(T)} \times \sigma_1^2(T) + \omega_{2(T)} \times \sigma_2^2(T), \text{ avec}$$

ω_i , la probabilité d'être dans la classe i,

σ_i , la variance de la classe i,

T, le seuil optimal

Ainsi, nous pouvons appliquer ces thresholds à notre image précédemment convertie afin d'obtenir le résultat suivant:

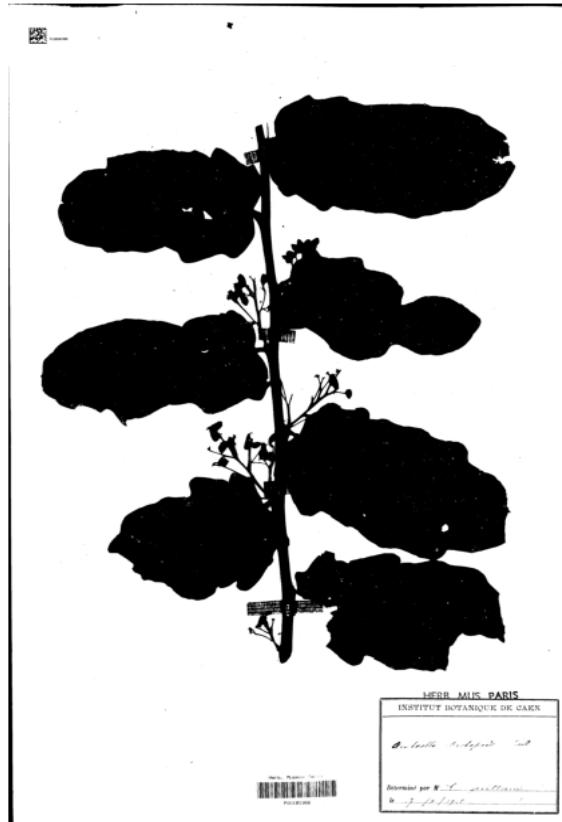


Figure 13. Image amborella075 binarisé avec Otsu

Nous pouvons remarquer deux phénomènes dans cette image étant que les attributs comme les étiquettes, écritures, codes-barres sont toujours visibles mais aussi que la distinction entre objets et fonds semble correctement prise en compte.

Toutefois, afin de permettre au threshold de considérer les attributs inutiles comme étant similaire au fond de l'image, nous avons appliqué au préalable un filtre Gaussien. Ce filtre correspond à un floutage de l'image de sorte à pénaliser uniquement certaines zones de l'image grâce à un kernel.

Ainsi, nous obtenons après l'utilisation du filtre Gaussien et l'application des thresholds, l'image suivante:



Figure 14. Image amborella075 avec un filtre Gaussien et binarisé avec Otsu

Sur cette image, nous pouvons observer un net progrès dans la suppression des attributs inutiles, bien qu'il reste du bruit notamment causé par les bordures des attributs.

Enfin, les dernières étapes du traitement correspondent à la recolorisation de l'image, son rognage et redimensionnement. La recolorisation de l'image a été effectuée en utilisant un masque des valeurs des pixels de l'image originale.

Le rognage est une partie importante du traitement car il influe sur le contenu visible de l'herbier. Il s'agit d'une étape importante afin de conserver les ratios de l'image (on peut constater sur les images des herbiers qu'il représente un rectangle). Les étapes du rognage sont les suivantes:

- Calcul du ratio hauteur/largeur.

Si le ratio est plus grand que 1 alors le rognage s'applique sur la hauteur sinon la largeur.

- Calcul du nombre de pixels à rogner.

Ce nombre est divisé par 2 afin de considérer un rognage centré.

- Détermination de la limite des pixels.

Après analyse de différentes images, nous avons remarqué que les images ont un ratio hauteur supérieur mais aussi que beaucoup de bruit se retrouvait visible en bas de l'image. Ainsi, nous avons fait un choix arbitraire dans notre cas de 75% en haut et 125% en bas.

Enfin, après avoir rogné l'image, nous pouvions ainsi la redimensionner. Nous avons fait le choix d'image de taille (512, 512) car cette taille permet de conserver la visibilité des caractères descriptifs des végétaux tout en permettant d'augmenter la rapidité du preprocessing et de la modélisation. Après avoir réalisé ces trois étapes, nous obtenons la figure suivante:



Figure 15. Image amborella075 après traitement complet

Nous pouvons remarquer que l'image n'a que très peu de bruits et la majeure partie des attributs inutiles sont supprimés. Ainsi, cette image représente le résultat final d'après traitement.

Par la suite, nous transformons l'entière des images de manières automatiques afin de pouvoir commencer la modélisation et l'apprentissage.

c. Apprentissage

Afin d'obtenir les meilleures performances lors de l'apprentissage, nous entraînons le modèle avec 3 learning rates différents sur 15 epochs. À partir de ces entraînements, nous pouvons afficher les loss curves et accuracy pour chacun caractères descriptifs.

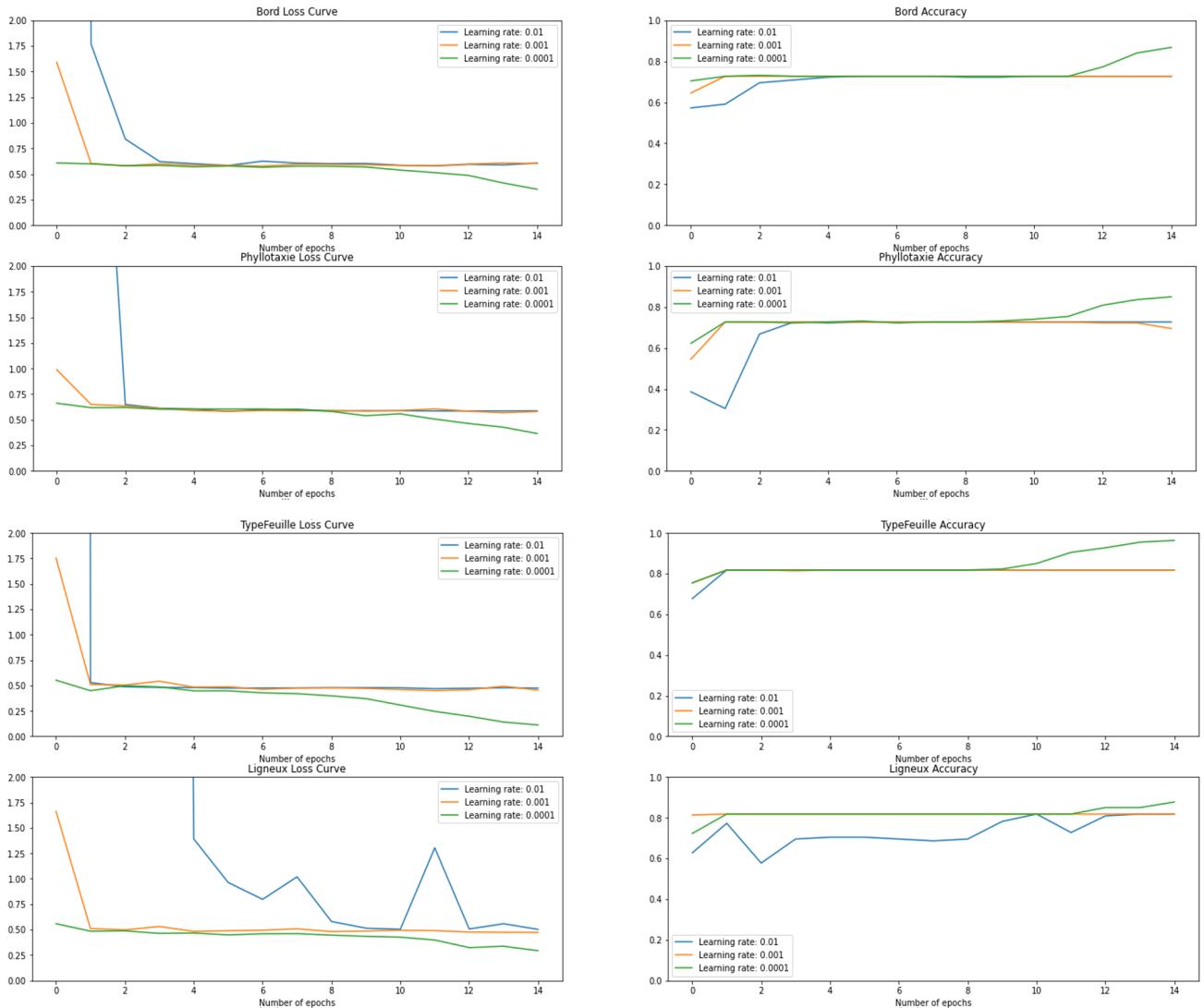


Figure 16. Loss curve et Accuracy en fonction du learning rate

On peut voir sur la figure 16 les courbes de précisions et de pertes en fonction du learning rate, on remarque que la précision atteint environ dépasse les 80% sur les 4 modèles permettant de déterminer les caractéristiques des herbiers et une précision très faible sur le modèle permettant de déterminer l'espèce de la plante dû à la faible quantité de données disponibles pour chaque espèce. On remarque qu'avec un learning rate de 0.0001 qui est représenté par les courbes vertes on atteint une précision bien plus élevée et une perte bien plus faible. On choisit donc un learning rate de 0.0001 pour entraîner le modèle.

Une fois le learning rate choisi, nous pouvons entraîner ce modèle avec celui-ci. Nous obtenons ainsi les figures suivante concernant le training et validation loss:

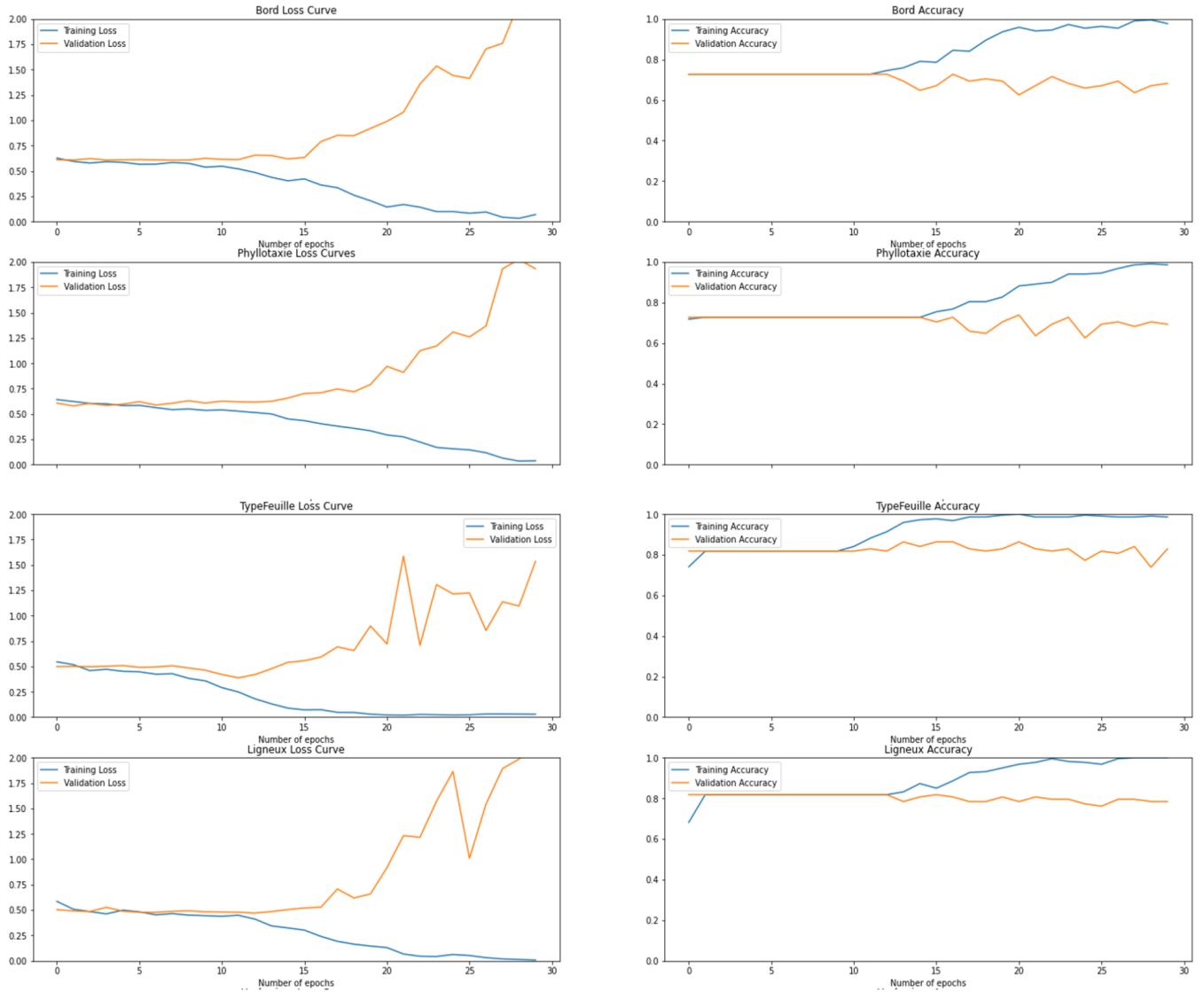


Figure 17. Loss curve et Accuracy sur le training dataset et validation dataset

On peut voir sur la figure 17 la précision et la perte des différents modèles avec un learning rate de 0.0001. On voit que la training accuracy est proche de 1 tandis que la validation accuracy est proche de 0.8, ce qui est une précision correcte étant donné la taille du jeu de données. On remarque un écart assez important entre la précision d'entraînement et de validation, la précision d'entraînement est bien plus élevée ce qui peut indiquer qu'il y a du sur-apprentissage avec ce modèle.

Les résultats montrant un clair sur-apprentissage, nous avons décidé de mettre en place de la data augmentation pour essayer d'obtenir de meilleures performances avec notamment de la rotation:

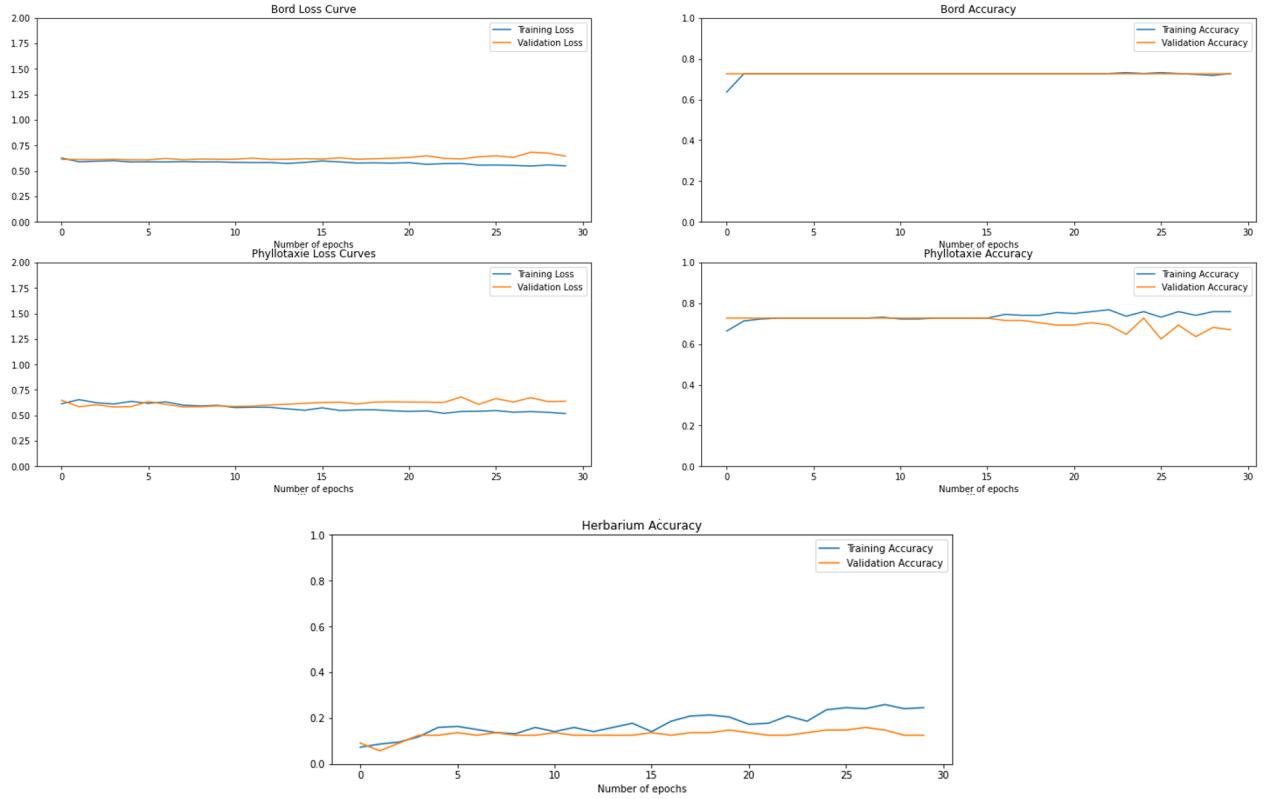


Figure 18. Loss curve et Accuracy sur le training dataset et validation dataset après data augmentation

On peut voir sur la figure 18 les courbes de précisions et de pertes avec le learning rate choisi. On voit que les écarts entre les courbes d'entraînement et de la validation sont bien plus petits ce qui indique que la modèle fait beaucoup moins d'overfitting sur le jeu de données après data augmentation et que la précision est proche de 75%.

Nous pouvons regarder la table ci dessous qui montre les différentes métriques calculés pour chacun des caractères descriptifs des plantes:

	Bord	Phyllotaxie	TypeFeuille	Ligneux	Herbarium
Accuracy score	0.73	0.73	0.82	0.82	0.05
Precision score	1	0.33	1	1	0.05
Recall score	0.73	0.5	0.82	0.82	0.02
F1 score	0.84	0.4	0.9	0.9	0.03

Table 2. Comparaison des métriques sur chaque caractéristique.

Nous avons calculé l'accuracy score, le precision score, le recall score et le F1 score du modèle. Le score F1 est souvent utilisé comme une seule métrique pour résumer les performances d'un modèle de classification, car il prend en compte à la fois la précision et le rappel. Un modèle avec un score F1 élevé est capable de faire des prédictions positives précises et a un faible taux de faux positifs et de faux négatifs. On remarque dans le tableau (Table 2) un score F1 élevé sur les caractéristiques Bord, Type Feuille et ligneux et un score faible pour la phyllotaxie et l'espèce de l'herbier ce qui peut être expliqué par la taille du jeu de données.

Nous pouvons aussi observer la figure suivante qui montre les différentes matrices de confusion de notre modèle pour les différentes classifications:

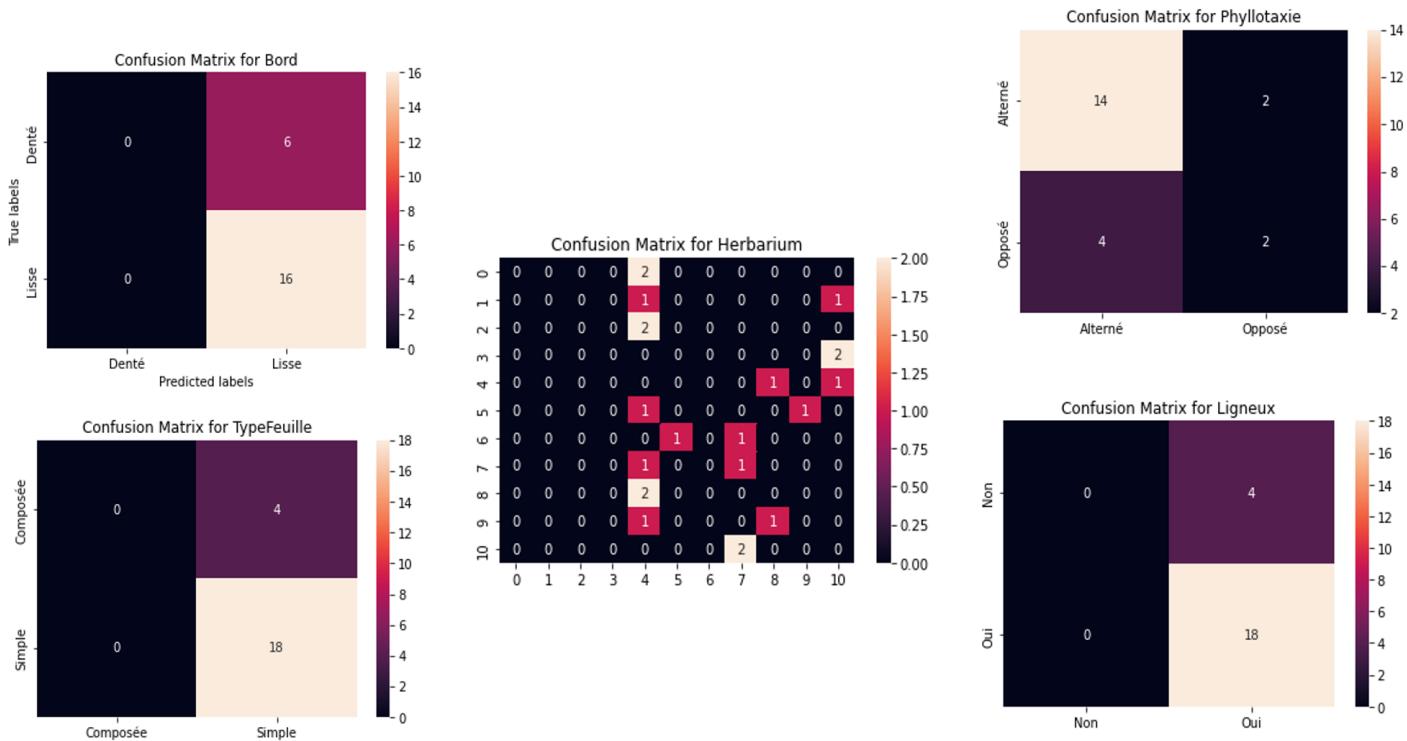


Figure 19. Matrices de confusion

Sur la figure ci-dessus, nous pouvons observer qu'il y a peu de faux positifs pour déterminer les caractéristiques des herbiers sur (matrice des côtés) mais beaucoup de mauvaises prédictions pour déterminer l'espèce des herbiers (matrices du milieu).

IV. Conclusion

Dans cette partie, nous allons conclure sur les méthodes et les résultats obtenus concernant le traitement des images ainsi que l'apprentissage de notre modèle.

a. Traitement de l'Image

Nous avons présenté les deux méthodes de traitement des images ainsi que leurs résultats associés.

Nous pouvons constater que la première méthode permet d'obtenir un meilleur nettoyage des images ainsi qu'une meilleure suppression des objets à retirer (étiquettes, etc.) cependant, celle-ci due à sa nécessité d'être modifiée manuellement ne peut être concevable sur un jeu de données réel ou sur un jeu de milliers d'images.

Ainsi, nous avons proposé une seconde méthode, permettant d'automatiser la nettoyage des images ainsi que respecter les différentes exigences comme l'aspect ratio. Cette méthode bien que moins efficace que la première permet toutefois d'être utilisée sur un grand jeu de données.

b. Apprentissage

En traçant la loss curve et l'accuracy en fonction de nos 3 learning rates différents, nous pouvons comparer l'influence de chaque learning rate sur la convergence et l'efficacité de l'entraînement du modèle. Plus précisément, nous pouvons observer les variations de la perte au fil des epochs et identifier les tendances et les comportements du modèle.

On remarque sur la figure 11 que pour un learning rate de 0.01, le learning rate est trop élevé ou par rapport aux caractéristiques du modèle et du dataset étant donné qu'il oscille de manière importante et ne converge pas.

Nous choisissons alors un learning rate de **0.0001**, car il est beaucoup plus stable, converge plus rapidement, donne une meilleure accuracy et minimise la perte plus efficacement.

Ensuite, en regardant la figure 12, on observe que les courbes de training et de validation divergent au fil des epochs.

Cela peut indiquer un phénomène d'**overfitting** (ou surapprentissage). L'overfitting se produit lorsque le modèle s'adapte trop spécifiquement aux données d'entraînement, et ne parvient pas à généraliser ses connaissances sur des données inconnues.

Pour réduire ce phénomène d'overfitting, nous avons utilisé des techniques de **data augmentation**.

La data augmentation est une technique qui consiste à générer de nouvelles données à partir des données existantes, en utilisant des transformations et des variations aléatoires.

La data augmentation peut être utilisée pour réduire l'overfitting dans certaines situations, en apportant de la variabilité et de la diversité aux données d'entraînement.

Sur la figure 13, on remarque que nous avons réduit de façon significative l'overfitting, ce qui montre l'efficacité de cette technique.

c. Difficultés Rencontrées

Tout au long du projet, nous avons fait face à certaines difficultés qu'il nous a fallu comprendre afin de les résoudre.

- La principale difficulté était de trouver une méthode afin de maximiser le traitement des images. Pour ce faire, nous avons utilisé l'ensemble de nos connaissances sur le sujet afin d'y tester les différentes méthodes, binarisation, étirement, égalisation ainsi que l'utilisation de nombreux filtres Gaussien, Prewitt, etc. Ceci nous a permis d'avoir le résultat qui nous semblait le plus fiable bien qu'imparfait.

- Nous avons aussi dû chercher à comprendre comment créer des données afin de pouvoir utiliser les images dans le modèle, pour ce faire, nous avons construit un dataframe train, test et validation avec comme information principale le chemin vers l'image et ces caractères descriptifs.

Toutefois, nous n'avons pas été capables de résoudre toutes les difficultés. Par exemple, une difficulté étant la possibilité de construire un réseau de convolutions ultra complexe avec un grand nombre de couches. Cependant, la demande en mémoire était élevée.

D'un point de vue moins technique, nous avons certains problèmes de communication et notamment dans la répartition des tâches.

V. Perspectives

Il existe de nombreuses améliorations possibles afin d'accroître les performances de notre modèle. Voici quelques exemples auxquelles nous avons pensé:

1. La modification de l'approche de conservation de l'aspect ratio des images avec l'implémentation d'une méthode de **padding**. La différence avec le rognage étant que l'ajout de padding permet de conserver l'entièreté du contenu des images.
2. L'amélioration du traitement de l'image avec l'application d'un squelette pour détecter les bordures des feuilles et conserver uniquement les objets végétaux.
3. Création d'un modèle de détection d'objet avec l'utilisation d'outils comme **Label-Studio** afin d'étiqueter les images ce qui permet l'accroissement du nombre de données disponibles à entraîner dans notre modèle.
4. Rechercher des données similaires aux herbiers mais externes aux modèles afin d'étudier son efficacité dans la compréhension des caractères descriptifs des végétaux.
5. Utilisation de modèles spécifiques pour la détection d'objet en 1-Stage comme YOLO ou SSD ou encore en 2-Stage comme Faster R-CNN.

Nous n'avons pas eu l'opportunité d'essayer ces pistes d'amélioration par manque de temps, cependant, il aurait été intéressant de s'y attarder afin de permettre une compréhension étendue des possibilités autour de ce projet.

En finalité de ce rapport, nous pouvons dire que nous nous sommes rendus compte de l'importance de comprendre et perfectionner son jeu de données ainsi que d'assurer que celui-ci soit suffisant et étudiabilie par le modèle. Il est impossible de construire un modèle sur un jeu de données que l'on ne maîtrise pas.