



DataScientest • com

Rapport Technique d'évaluation

Projet pyStill

Plant Expert at your finger

Promotion :

Data Scientist – Bootcamp Octobre 2020

<https://datascientest.com/>

Participants :

- ✓ **AXILAIS Loïc**
- ✓ **MAURICE Noël**
Ingénieur Informatique
noelmaurice4@gmail.com
- ✓ **RAVAILAO Mathis**

1. Contexte

Ce projet constitue une étape dans la validation de la certification à la formation « Data Scientist ».

Les participants se sont mobilisés sur ce projet pour les raisons suivantes :

✓ **AXIL AIS Loïc**

« Issue d'une formation agricole, ce projet de reconnaissance des plantes m'est apparu comme une évidence et un sujet d'intérêt dans le cadre de mon activité professionnelle.

Au-delà de l'aspect technique qui a permis d'aborder la reconnaissance d'images à l'aide du Deep Learning, l'intérêt principal du projet est de pouvoir m'approprier le fonctionnement d'un réseau de neurones, de l'incarner à travers un exemple concret, d'en comprendre les mécanismes et les limites.

Par ailleurs, mon bassin d'emploi (Rennes) est pourvoyeur d'activité dans le domaine de l'agriculture de précision (ex : société DILEPIX, salon du SPACE, ...). Ce projet est une bonne manière de démontrer mon savoir-faire sur ce sujet. »

✓ **MAURICE Noël**

« Le deep learning permet des avancées scientifiques fulgurantes, dans l'astronomie, la physique, le médical, l'agronomie et dans bien d'autres disciplines...

L'humanité doit aussi trouver des solutions aux problèmes qu'elle s'est créée. De nouveaux défis sont bel et bien présents.

Par exemple, la surpopulation et le changement climatique menacent la production alimentaire.

Dans ce domaine, le deep learning permet déjà aux agriculteurs de déployer des équipements capables de repérer et de différencier les plantes cultivées et les mauvaises herbes.

L'utilisation d'herbicides ou d'engrais peut donc être réduite et la production agricole améliorée : c'est un moindre mal dans un océan de pollution planétaire.

La communauté scientifique a assurément le regard tourné vers le deep learning et notamment dans le traitement d'images avec les réseaux neuronaux convolutifs (CNN).

Le projet pyStill permet tout à la fois d'expérimenter et d'implémenter des réseaux de neurones (RN) mais également d'utiliser ces RN pour la détection de maladies de plantes.

Tant pour la phase d'expérimentation scientifique, que pour la mise en application technique et l'approche professionnelle, ce projet prend donc toute sa place aujourd'hui.

Mon intérêt pour l'écologique et l'humanitaire motive aussi mon choix pour ce projet. »

✓ **RALAIVAO Mathis**

« *En tant qu'ingénieur chimiste, j'ai démarré cette formation de Data Scientist dans le but d'appliquer les technologies et méthodes apprises à des sujets sur lesquels je travaillais précédemment.*

L'étude des plantes m'intéresse énormément. J'ai travaillé sur le développement de matières premières naturelles pour remplacer des composés provenant de la pétrochimie.

A travers ce projet, je voulais voir la puissance des outils informatiques pour travailler sur les plantes. J'utilise déjà une application de reconnaissance d'espèces végétales, cette application est très efficace, je voulais donc en savoir plus sur son fonctionnement. »

2. Objectifs

2.1. Fonctionnels

Le projet « pyStill » propose de détecter d'éventuelles maladies sur les plantes.

Le principe est simple pour l'utilisateur, il suffit de fournir au système la photographie d'une feuille de la plante et celui-ci identifie la plante et indique si elle est saine.

Dans le cas contraire, le système indique la maladie détectée. Lorsqu'une maladie est détectée, le système propose quelques informations et conseils pour le traitement.

Nous nous sommes concentrés sur les plantes suivantes : Cerise et griotte, Courge, Fraise, Framboise, Maïs, Myrtille, Orange, Pêche, Poivron, Pomme, Pomme de terre, Raisin, Soja, Tomate.

2.2. Techniques

Bien que notre projet repose avant tout sur la connaissance scientifique, les concepts et les structures de données rencontrés durant la formation de « Data Scientist », notamment pour définir les algorithmes qui répondront à la problématique du projet, le langage python nous a permis d'implémenter la solution.

Nous avons donc utilisé les librairies python suivantes :

- Manipulation de fichiers : os, sys, glob ;
- Traitement multiprocesseur : threading, multiprocessing ;
- Manipulation d'images : cv2, PIL ;
- Manipulation de matrices : numpy, pandas ;
- Visualisation de données : matplotlib, seaborn ;
- Réduction de dimensions : sklearn (PCA) ;
- Réseau de neurones et prétraitement d'images : keras, tensorflow
 - ✓ Segmentation d'images : basée sur la structure du modèle U-Net ;
 - ✓ Reconnaissance d'images : basée sur une structure expérimentale puis sur une structure LeNet ;
 - ✓ Apprentissage par transfert : depuis un VGG16 pré-entraîné sur le jeu de données « imangenet ».

2.3. Pédagogiques

Le projet « pyStill » permet de mettre en œuvre une méthodologie applicable à la très grande majorité des projets informatiques relevant d'un problème d'apprentissage artificiel :

2.3.1. Sur le périmètre du projet

- Comprendre la demande du client (DataScientest) ;
- Identifier les sources d'informations disponibles, matière première de l'apprentissage, avec l'aide du client (qui est l'acteur du projet maîtrisant le mieux les données relevant de son domaine de compétence) ;
- Explorer les données et évaluer leur qualité ;

- Chercher d'autres sources de données pour compléter ou améliorer si besoin la qualité du jeu de données existant ;
- Sélectionner les données éligibles à l'entraînement d'un modèle ;
- Choisir un ou plusieurs modèles répondant à la problématique et les implémenter ;
- Evaluer la qualité du ou des modèles ;
- Réajuster le ou les modèles pour les optimiser.

2.3.2. Sur le pilotage du projet

- Répartir les tâches dans l'équipe en fonction des compétences et motivations de chacun ;
- Suivre le planning d'avancement des tâches et évaluer les résultats ;
- Réaliser des points réguliers avec le client représenté par Thomas BOEHLER, employé « DataScientest », pour s'assurer de l'adéquation de la réponse à sa demande ;
- Organiser la production en livrable à travers une interface graphique et ce document ;
- Présenter le travail réalisé et démontrer le respect de la réponse à la demande.

3. Niveau d'expertise sur la problématique adressée

L'équipe est constituée de profils divers, chaque membre de l'équipe disposait avant la formation de certaines connaissances en relation avec le projet :

✓ **AXILAIIS Loïc**

« *D'un point de vue technique, je ne connaissais que ce qui avait été abordé en formation mais d'un point de vue fonctionnel et organisationnel, j'étais plutôt en terrain connu.*

La reconnaissance de maladies sur les plantes est un sujet aussi vieux que l'agriculture et j'avais déjà œuvré sur un projet similaire (sans algorithme d'IA) avec une reconnaissance de parasites sur les cultures à l'aide de lunettes connectées (Vuzix).

Evidemment, l'apprentissage artificiel apporte une nouvelle dimension au sujet. La capacité d'un système à reconnaître une plante et une maladie sans l'assistance humaine offre de nouvelles perspectives »

✓ **MAURICE Noël**

« *Avant d'intégrer la formation de DataScientist, j'avais suivi plusieurs cours sérieux sur le net, notamment sur le Machine Learning et de manière plus généraliste sur les RN.*

Mais en règle générale, les formations du Net, même sérieuses, reste assez exhaustives et n'ont pas, ou peu, de finalité professionnelle.

Déjà par le domaine d'application du Projet, mais également par les méthodologies IA et Deep Learning que je souhaitais approfondir, ce projet prenait donc tout son sens, tant pour la partie scientifique et théorique, que pour la partie mise en œuvre et l'approche professionnelle.

Pour la connaissance métier, tout restait à faire. En effet, en tant qu'Ingénieur en Informatique plus spécialisé dans le domaine de la Recherche et de la Formation ainsi que dans le Génie Logiciel, mon métier ne m'a jamais orienté vers le domaine de l'agronomie ou de l'agriculture. »

✓ **RALAIVAO Mathis**

« *De part une utilisation récurrente d'une application de reconnaissance d'espèces végétales, j'ai pu me mettre dans la peau de l'utilisateur pour notre projet. Il a été ainsi plus facile de comprendre les attentes de ce point de vue.*

Pour la partie technique, je n'avais jamais travaillé sur un projet de classification d'images par Machine Learning mais les cours de la formation m'ont permis de me mettre rapidement à niveau et pouvoir comprendre, implémenter et utiliser un réseau de neurones efficace pour classer des familles de plantes. »

4. Données

4.1. Cadre

Nous avons étudié un ensemble de jeux de données libres de droits, jeux de données proposés par « Datascientest » pour ce projet. Ils sont présentés ci-après :

4.1.1. COCO Common Objects in Context

Source : <https://cocodataset.org/#home>

Le site propose un ensemble de données labellisées (véhicules, animaux, mobiliers, ...) mais il n'y a rien d'utilisable sur les plantes (à part quelques plantes en pot).

Par contre, le site présente le principe de détection d'objet et de segmentation dont nous nous sommes inspirés dans le cours du projet.

Le site reste intéressant pour tester notre modèle sur des photos ne représentant pas des plantes afin de vérifier comment il se comporte.

4.1.2. Open Image Dataset

Source : <https://storage.googleapis.com/openimages/web/download.html>

Le site propose un vaste ensemble de données labellisées (près de 20 000 classes) ainsi que des jeux de données avec "Bounding Box" et masque de segmentation (600 classes).

Comme le COCO Dataset, il n'y a rien d'exploitable concernant les maladies des plantes.

4.1.3. Kaggle

Identification d'espèces

Source : <https://www.kaggle.com/vbookshelf/v2-plant-seedlings-dataset> (3,2 Go)

Source : <https://www.kaggle.com/miljan/plantclef-2019-amazon-rainforest-plants-images> (0,4 Go)

Reconnaissance de maladies

Source : <https://www.kaggle.com/vipooooool/new-plant-diseases-dataset> (1,5 Go)

Source : <https://www.kaggle.com/saroz014/plant-disease> (1,8 Go)

Source : <https://www.kaggle.com/abdallahalidev/plantvillage-dataset> (2,3 Go)

Les jeux de données sur Kaggle semblent particulièrement adaptés à notre problématique.

Nous nous sommes concentrés dessus à travers l'exploration présentée dans la suite de ce document.

4.1.4. Plant Village

Source : <https://plantvillage.psu.edu>

Le site dispose d'une base de connaissances sur les maladies des plantes que nous pouvons valoriser.

Une fois la reconnaissance par prédition de la plante et de la maladie effectuée par notre système, les informations de ce site sont reprises pour renseigner l'utilisateur sur la plante détectée.

4.2. Pertinence

Nous avons entraîné le modèle d'identification sur le jeu de données [new-plant-diseases-dataset](#).

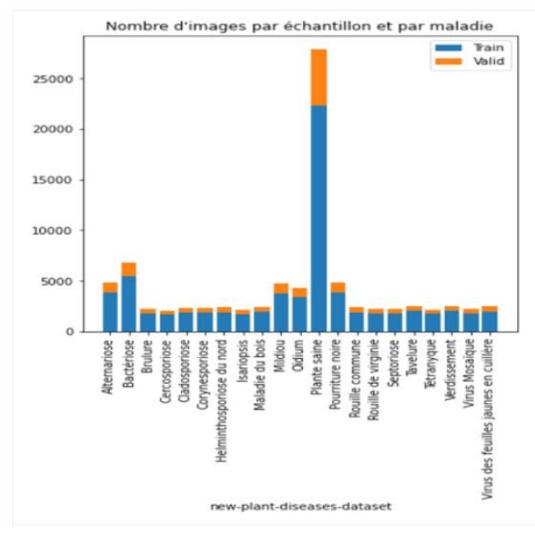


Figure 1 - Répartition des photos par classe

Les données utilisées sont de bonne qualité : homogène, réparties de manière équilibrée entre classes, de même dimension, organisées dans le format attendu par keras.

Remarque : la classe « Plante saine » du graphique est sur-représentée de manière artificielle car elle regroupe les images de toutes les plantes saines du jeu de données (cerise + fraise + framboise, etc.).

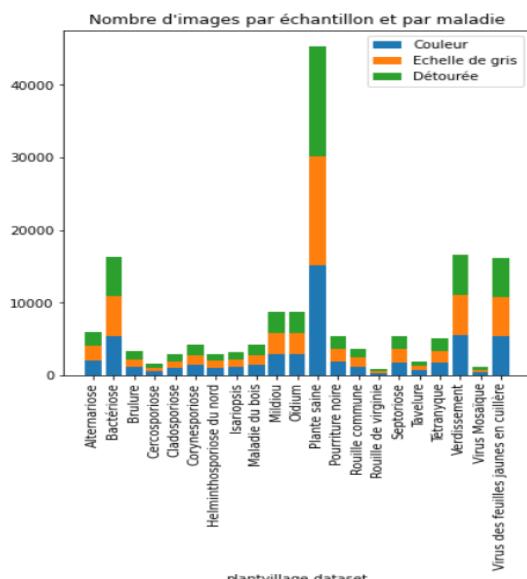


Figure 2 - Répartition des photos par couleur

Par ailleurs, nous avons entraîné le modèle de segmentation sur le jeu de données **plantvillage-dataset** qui proposait les mêmes classes que **new-plant-diseases-dataset** mais avec des images déjà détournées (voir graphe ci-contre).

Remarque : la classe « Plante saine » du graphique est sur-représentée de manière artificielle car elle regroupe les images de toutes les plantes saines du jeu de données (cerise + fraise + framboise, etc.).

Nous n'avons pas eu besoin de faire de « nettoyage » dans les données car elles sont de bonnes factures et ont une résolution graphique très correcte pour la finalité du projet.

Nous pouvons toutefois relever deux défauts :

1. Toutes les photos de feuilles sont **centrées** comme le montre le calcul de l'image moyenne de chaque classe dont un extrait est présenté ci-dessous, ce qui peut constituer un biais.

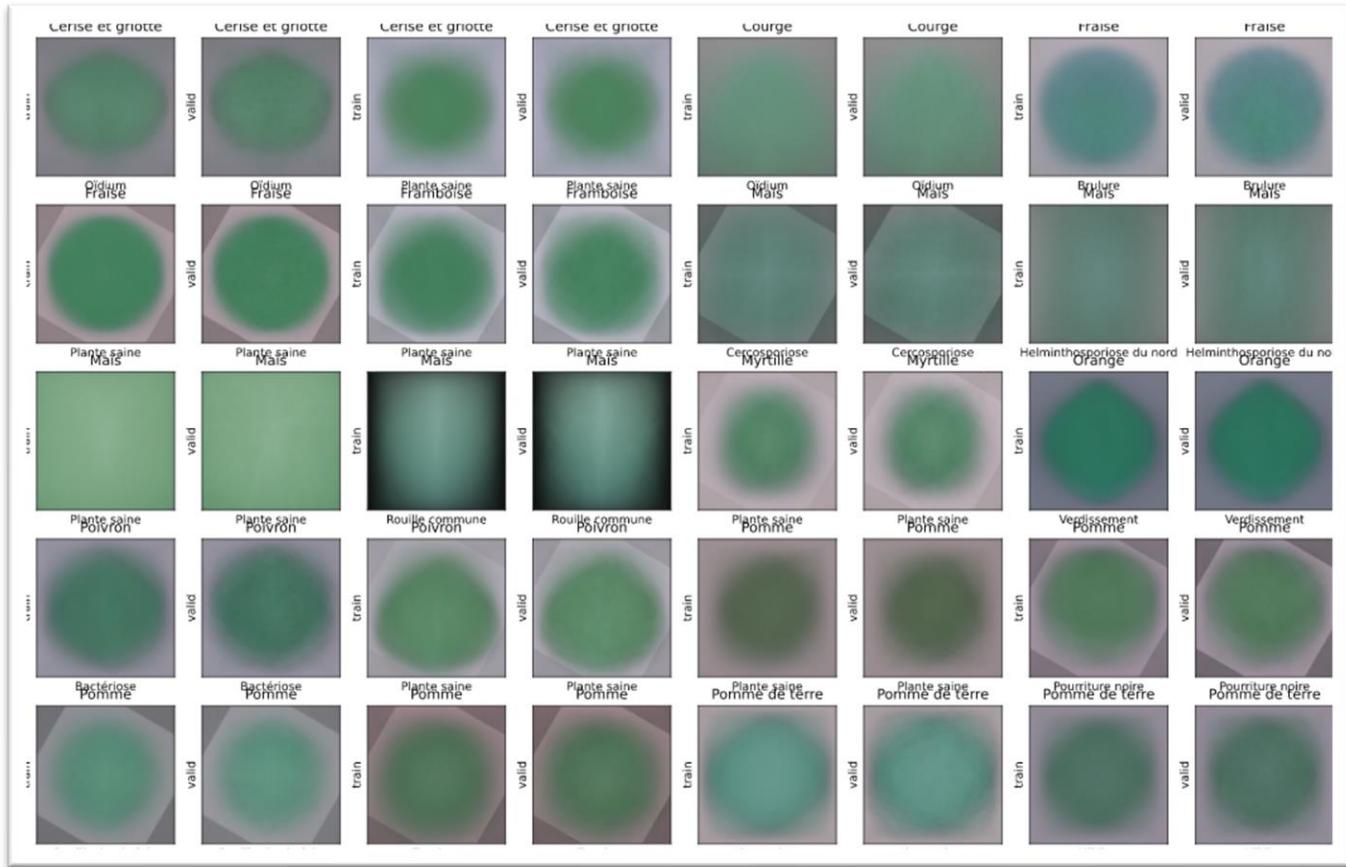


Figure 3 - Image moyenne de chaque classe

2. Le nombre d'images par classe n'est pas très élevé (un peu moins de 2000) comme le montre le graphique ci-dessous :

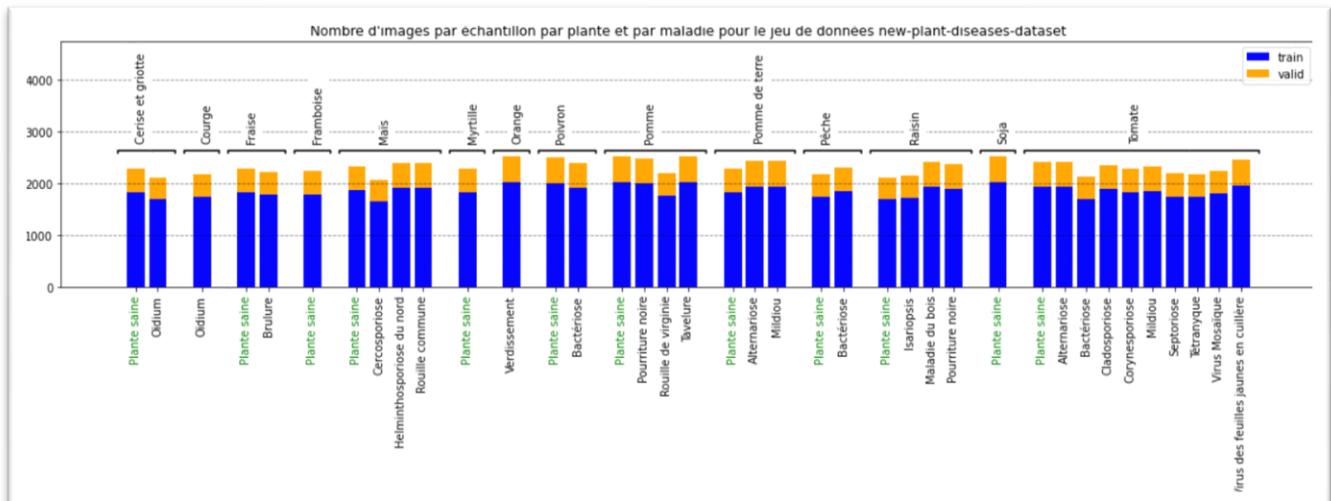


Figure 4 - Nombre d'images par classe

Mais en soi, cela ne devrait pas poser de soucis majeurs. Pour remédier à ces 2 points, l'augmentation de données pour ajouter des images et appliquer des transformations (notamment des décalages en hauteur et en largeur) avant l'entraînement des modèles devrait être efficace.

Grace à ce jeu de données, dont les images ont pour résolution graphique (256, 256, 3), le modèle pourrait reconnaître 38 classes, recensées ci-après :

| CLASSE | PLANTE | MALADIE |
|--------------------------------------------------|-------------------|---------------------------------------|
| Apple_Apple_scab | Pommier | Tavelure |
| Apple_Black_rot | | Pourriture noire |
| Apple_Cedar_apple_rust | | Rouille de virginie |
| Apple_healthy | | Aucune |
| Blueberry_healthy | Myrtille | Aucune |
| Cherry_(including_sour)_healthy | Cerise et griotte | Aucune |
| Cherry_(including_sour)_Powdery_mildew | | Oïdium |
| Corn_(maize)_Cercospora_leaf_spot_Gray_leaf_spot | Maïs | Cercosporiose |
| Corn_(maize)_Common_rust_ | | Rouille commune |
| Corn_(maize)_healthy | | Aucune |
| Corn_(maize)_Northern_Leaf_Blight | | Helminthosporiose du nord |
| Grape_Black_rot | Vigne | Pourriture noire |
| Grape_Esca_(Black_Measles) | | Maladie du bois |
| Grape_healthy | | Aucune |
| Grape_Leaf_blight_(Isariopsis_Leaf_Spot) | | Isariopsis |
| Orange_Haunglongbing_(Citrus_greening) | Oranger | Verdissement |
| Peach_Bacterial_spot | Pêcher | Bactériose |
| Peach_healthy | | Aucune |
| Pepper,_bell_Bacterial_spot | Poivron | Bactériose |
| Pepper,_bell_healthy | | Aucune |
| Potato_Early_blight | Pomme de terre | Alternariose |
| Potato_healthy | | Aucune |
| Potato_Late_blight | | Mildiou |
| Raspberry_healthy | Framboisier | Aucune |
| Soybean_healthy | Soja | Aucune |
| Squash_Powdery_mildew | Courge | Oïdium |
| Strawberry_healthy | Fraisier | Aucune |
| Strawberry_Leaf_scorch | | Brûlure |
| Tomato_Bacterial_spot | Tomate | Bactériose |
| Tomato_Early_blight | | Alternariose |
| Tomato_healthy | | Aucune |
| Tomato_Late_blight | | Mildiou |
| Tomato_Leaf_Mold | | Cladosporiose |
| Tomato_Septoria_leaf_spot | | Septoriose |
| Tomato_Spider_mites_Two-spotted_spider_mite | | Tétranyque |
| Tomato_Target_Spot | | Corynesporiose |
| Tomato_Tomato_mosaic_virus | | Virus Mosaïque |
| Tomato_Tomato_Yellow_Leaf_Curl_Virus | | Virus des feuilles jaunes en cuillère |

Suite à ce qui précède et à notre analyse plus poussée des données, analyse reprise en annexe, nous avons démarré le projet en utilisant le jeu de données [new-plant-diseases-dataset](#).

Ce document expliquera par la suite pour quelles raisons il a été nécessaire d'enrichir ce jeu de données, mais seulement pour l'ajout d'un complément fonctionnel.

5. Projet

5.1. Classification du problème

Ce projet est avant tout un problème de classification avec reconnaissance d'images.

Mais autant il est nécessaire d'identifier le contenu d'une image, autant il reste intéressant d'isoler ce contenu dans l'image.

Bien souvent une image contient plusieurs objets et avant l'identification d'un objet particulier il est nécessaire de l'isoler du reste du contenu de l'image.

Même si cela n'était pas spécialement obligatoire ici, la partie du projet dédié à la « segmentation » puis la partie « bounding box » permettent d'extraire chaque feuille du reste de l'image.

Ce projet s'articule donc autour de plusieurs problématiques IA qui sont :

- Classification des images ;
- Segmentation des images ;
- Extraction de la composante principale de l'image, soit une feuille, par une identification de la surface et de la position de cette composante par la technique de « bounding box ».

En plus de proposer des solutions avec leur implémentation à ces problématiques, il a été inclus au projet les parties suivantes :

- Etude de la réduction de dimension sur les images : par la suite, il n'a pas été nécessaire dans le projet de mettre en œuvre ce concept.
Ce rapport ne présentera donc pas cette étude qui est intégrée aux annexes de ce document.
- Réalisation d'une interface graphique de présentation du projet et de démonstration.
Ce rapport ne discutera pas de cette interface qui reste disponible en ligne en complément à ce document.
Au moment de la finalisation du rapport, l'adresse internet vers cette interface graphique n'est pas définitive.
Il convient donc de s'adresser aux auteurs pour obtenir cette adresse internet.

5.2. Partie principale du projet

La partie principale du projet est la classification des plantes et l'identification des maladies. Pour résoudre cela, nous avons mis en œuvre des techniques de Deep Learning en nous appuyant sur plusieurs réseaux de neurones :

- Réseau de neurones classique servant avant tout de « baseline » au projet ;
- Réseau de neurones amélioré et basé par le CNN LeNet ;
- Méthode de « transfer learning » en utilisant un CNN de type VGG16.

Pour la partie classification, nous nous sommes intéressés principalement à deux métriques :

- La fonction de perte à minimiser ('categorical_crossentropy') qui est utilisée pour les modèles de classification de plusieurs catégories ;
- La performance à maximiser ('accuracy') qui est utilisée comme méthode de calcul de la performance du modèle.

Même si plusieurs itérations ont été utilisées au cours du projet, les mêmes métriques ont été conservées pour tous les modèles de classification à des fins de comparaison.

Pour la partie segmentation, nous nous sommes intéressés principalement à deux métriques :

- La fonction de perte à minimiser ('loss dice') ;
- La performance à maximiser ('accuracy').

6. Choix du modèle & Optimisation

Tout d'abord il est important d'indiquer que le projet pouvait être découpé en plusieurs parties, chacune dépendante des résultats du moment mais chacune demandant une solution différente.

Ces parties sont les suivantes :

- Classification des images ;
- Segmentation des images ;
- Extraction des objets « plantes » par la méthode « bounding box ».

Puisqu'il est question ici du « choix du modèle & l'optimisation », ce chapitre ne présentera pas les parties secondaires et nécessaires au projet, soit :

- Exploration des données et « Dataviz' » ;
- Réalisation de l'interface graphique de présentation des résultats du projet.

Pour information, les annexes reprennent les résultats issus de « l'exploration des données » et des algorithmes utilisés tout le long du projet pour les parties « classification », « segmentation » et « bounding box ».

Le module « interface graphique » sera par la suite disponible en ligne et au plus tard lors de la soutenance de ce projet.

Pour revenir au choix des modèles et à l'optimisation, voici ci-après les méthodologies et techniques qui ont été choisies.

6.1. Classification des images

Cette partie du projet a été itérative avec un démarrage très expérimental (voir partie sur les difficultés rencontrées lors du projet).

De par la force des choses, mais aussi par conviction scientifique, il restait très intéressant de respecter la chronologie : « Hypothèse » puis « Expérimentation » puis « Conclusion ».

Ce sont donc la somme des itérations qui conforte la conclusion de ce projet.

6.1.1. CNN classique et « baseline »

Pour commencer le projet, il était important d'expérimenter un premier réseau de neurones classique. Il était nécessaire d'avoir une base de référence pour l'implémentation du CNN final.

Certes, un modèle de classification classique comme par exemple RandomForest pourrait servir de « baseline » à ce projet. Néanmoins, devant la très grande quantité de photos disponibles dans le jeu de données, une implémentation basique d'un CNN permettra également d'avoir une « baseline » intéressante pour ce projet.

Donc cette itération permettra de définir un CNN très basique mais adapté au jeu de données disponible.

```

Model: "sequential"
-----  

Layer (type)           Output Shape        Param #
-----  

conv2d (Conv2D)        (None, 126, 126, 64) 1792  

max_pooling2d (MaxPooling2D) (None, 63, 63, 64) 0  

flatten (Flatten)      (None, 254016)       0  

dense (Dense)          (None, 128)          32514176  

dense_1 (Dense)         (None, 38)           4902  

-----  

Total params: 32,520,870  

Trainable params: 32,520,870  

Non-trainable params: 0

```

Figure 5 - CNN classique servant de baseline

Ce CNN a donc été entrainé et les scores ont pu être graphiquement visualisés.

Ce premier CNN a permis d'avoir des résultats qui ont rapidement convergé vers des scores assez satisfaisants, plus pour le jeu d'entraînement que pour le jeu de test : un « overfitting » ou surentrainement semblait être présent tout du long du déroulé de l'entraînement.

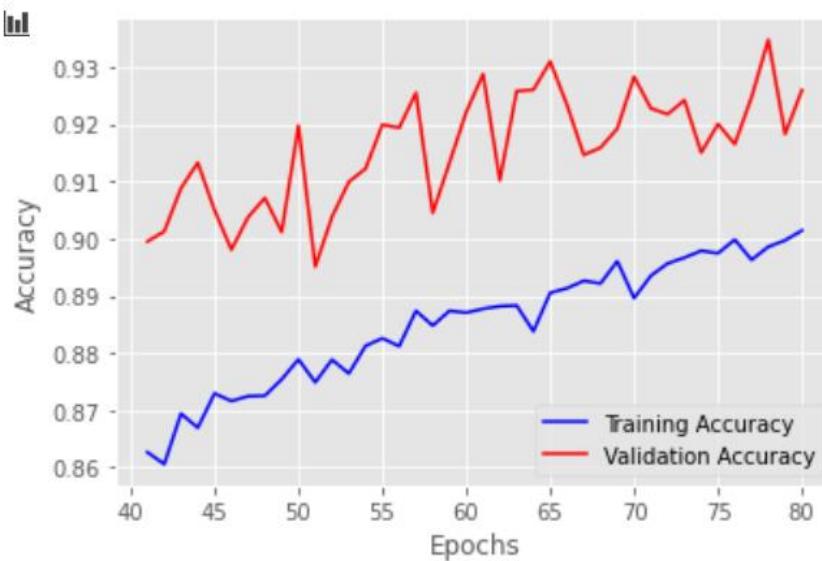


Figure 6 - CNN classique : courbe de performance et overfitting

Le temps d'entraînement choisi alors, et même s'il a pris beaucoup de temps, s'est avéré insuffisant pour atteindre des résultats satisfaisants.

Les valeurs de performances dépassaient les 80% pour le jeu de test, mais la courbe d'apprentissage semblait vouloir continuer sa progression au-delà du nombre des époques utilisées.

Ces résultats pouvaient servir de « baseline » : la suite du projet permettra l'exploration de solutions d'amélioration des résultats.

6.1.2. Architecture basée sur CNN LeNet

6.1.2.1. Elaboration du CNN

Au vu des expérimentations précédentes, un CNN intégrant les connaissances acquises et possédant une architecture plus adaptée aux objectifs a ensuite été implémenté.

En effet, il fallait que le nouveau modèle gagne en performance et aussi ne fasse plus de surentrainement ou « overfitting ».

L'architecture choisie, notamment à cause des ressources systèmes du moment disponibles, a été similaire à celui d'un CNN de type LeNet.

Le CNN précédent a donc été amélioré :

- une couche de Convolution a été ajoutée
- une couche de Max pooling a été ajoutée
- une couche Dense a été ajoutée
- une couche Dropout a été ajoutée pour éviter l'overfitting
- les hyper-paramètres ont été adaptés à ce nouveau CNN pour tenir compte du format des images en entrée soit une dimension de (256, 256, 3).

| Model: "sequential" | | |
|---------------------------------------------|----------------------|----------|
| Layer (type) | Output Shape | Param # |
| conv2d (Conv2D) | (None, 124, 124, 64) | 4864 |
| max_pooling2d (MaxPooling2D) | (None, 62, 62, 64) | 0 |
| conv2d_1 (Conv2D) | (None, 60, 60, 128) | 73856 |
| max_pooling2d_1 (MaxPooling2 (MaxPooling2D) | (None, 30, 30, 128) | 0 |
| flatten (Flatten) | (None, 115200) | 0 |
| dense (Dense) | (None, 160) | 18432160 |
| dropout (Dropout) | (None, 160) | 0 |
| dense_1 (Dense) | (None, 96) | 15456 |
| dropout_1 (Dropout) | (None, 96) | 0 |
| dense_2 (Dense) | (None, 38) | 3686 |

Total params: 18,530,022

Figure 7 - Structure CNN type LeNet

Les ressources système sur le serveur de « Datascientest », bien que supérieures aux ressources locales, restaient aussi limitées.

Il a été donc nécessaire de reprendre le code afin que le CNN puisse être entraîné correctement sur la plateforme « Google colab » (voir le chapitre sur les difficultés rencontrées lors du projet).

6.1.2.2. Optimisation du CNN

Une fois que la courbe de performance a convergé vers un maximum et s'est stabilisée vers cette valeur, une dernière itération d'entraînement a été réalisée avec un learning rate plus petit : cela a permis de s'assurer du score maximum du CNN.

Le maximum de ce modèle a été atteint : les valeurs des dernières époques d'entraînement sont devenues très stables et l'écart type des distributions, pour l'entraînement et la validation, était extrêmement faible.

Cela a été principalement dû à l'ajustement de l'hyper-paramètre learning rate qui a été choisi de plus en plus petit au fur et à mesure de la convergence des courbes d'apprentissage vers leur maximum.

6.1.2.3. Résultats du CNN

Les résultats de performance du CNN ont été de :

- près de 97,3 % pour le jeu de données d'entraînement
- près de 98 % pour le jeu de validation

Les courbes d'entraînement et de test étaient quasi-superposées ce qui montre que le modèle ne fait ni d'overfitting, ni d'underfitting.

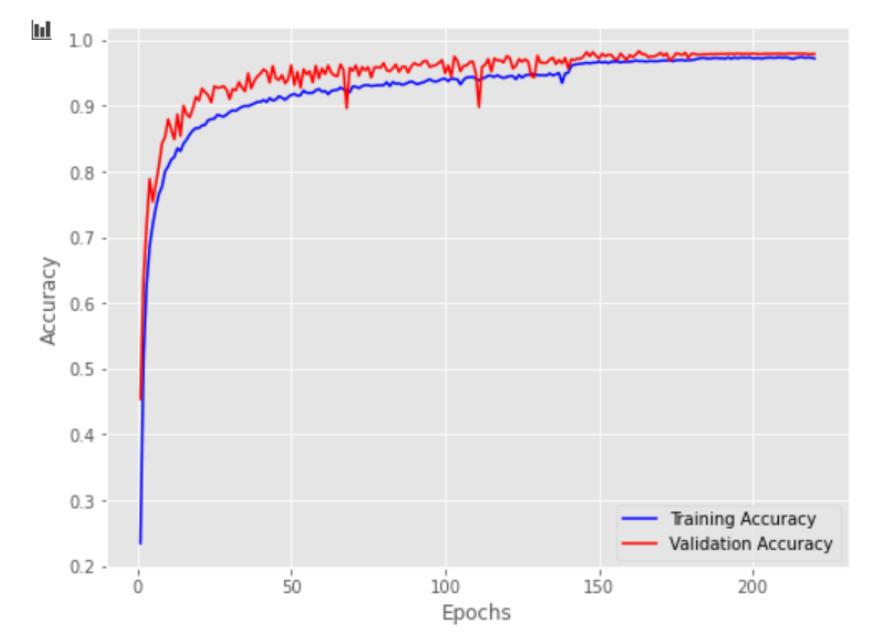


Figure 8 - CNN type LeNet - Courbe de performance

Sur le jeu de données de test, la prédiction du CNN a été faite avec 100 % de réussite. Le moins bon indice de confiance des prédictions a été de près de 92 %, et une très grande majorité des prédictions a été réalisée avant un indice de confiance compris entre 99 % et 100 %.

La performance du CNN était maintenant très satisfaisante et excellente.

6.1.2.4. Conclusions intermédiaires

A ce stade du projet, il était raisonnable de pouvoir dire que les objectifs étaient atteints.

Bien entendu, un CNN avec plus de couches de Convolution + Max pooling + Dense + Dropout, et des images d'entrée de dimension (256, 256, 3) permettrait peut-être d'améliorer de quelques dixièmes de point la performance, voire d'approcher le score summum de 100% : mais pour quels coûts de calculs, de temps, de ressources système...

Aussi lors de la formation « DataScientest », la méthode de « transfer learning » venait d'être présentée.

Malgré les performances très satisfaisantes du CNN et voulant expérimenter d'autres pistes d'amélioration à moindre coût « système », cette piste fut ensuite suivie, ce que les paragraphes suivants vont présenter.

6.1.3. Transfer Learning et VGG16

Conformément au cours relatif aux CNN, la méthode de « transfer learning » fut appliquée en utilisant un CNN VGG16 et en l'entraînant, dans un premier temps, uniquement sur sa partie classification, puis ensuite en poursuivant son entraînement sur ses dernières couches de convolution et la partie classification.

Ce CNN a été initialisé sur le jeu de données « imangenet ».

| Model: "sequential_1" | | |
|----------------------------------|-------------------------|----------|
| Layer (type) | Output Shape | Param # |
| vgg16 (Functional) | (None, None, None, 512) | 14714688 |
| global_average_pooling2d_1 (| (None, 512) | 0 |
| dense_3 (Dense) | (None, 1024) | 525312 |
| dropout_2 (Dropout) | (None, 1024) | 0 |
| dense_4 (Dense) | (None, 512) | 524800 |
| dropout_3 (Dropout) | (None, 512) | 0 |
| dense_5 (Dense) | (None, 38) | 19494 |
| <hr/> | | |
| Total params: 15,784,294 | | |
| Trainable params: 1,069,606 | | |
| Non-trainable params: 14,714,688 | | |

Figure 9 - VGG16 – Entrainement sur partie classification

| Model: "sequential_1" | | |
|---------------------------------|-------------------------|----------|
| Layer (type) | Output Shape | Param # |
| vgg16 (Functional) | (None, None, None, 512) | 14714688 |
| global_average_pooling2d_1 (| (None, 512) | 0 |
| dense_3 (Dense) | (None, 1024) | 525312 |
| dropout_2 (Dropout) | (None, 1024) | 0 |
| dense_4 (Dense) | (None, 512) | 524800 |
| dropout_3 (Dropout) | (None, 512) | 0 |
| dense_5 (Dense) | (None, 38) | 19494 |
| <hr/> | | |
| Total params: 15,784,294 | | |
| Trainable params: 8,149,030 | | |
| Non-trainable params: 7,635,264 | | |

Figure 10 - VGG16 – Entrainement sur partie classification et couches de convolution supérieures

A partir de ce moment du projet et toujours en utilisant la plateforme « Google colab », il a été possible d'entraîner le nouveau CNN sur des images de dimension (256, 256, 3) au lieu de la dimension (126, 126, 3) utilisée précédemment.

Le CNN a rapidement convergé vers des scores dépassant les 98% puis après optimisation a largement dépassé les 99% de performance, tant pour les données d'entraînement que pour les données de validation.

Donc avec un entraînement jusqu'à près de 6 fois inférieur au CNN classique, soit environ 40 époques au lieu des 220 époques pour le CNN classique, le VGG16 a atteint des scores supérieurs à ce dernier CNN qui pourtant avait de très bons scores de performance.

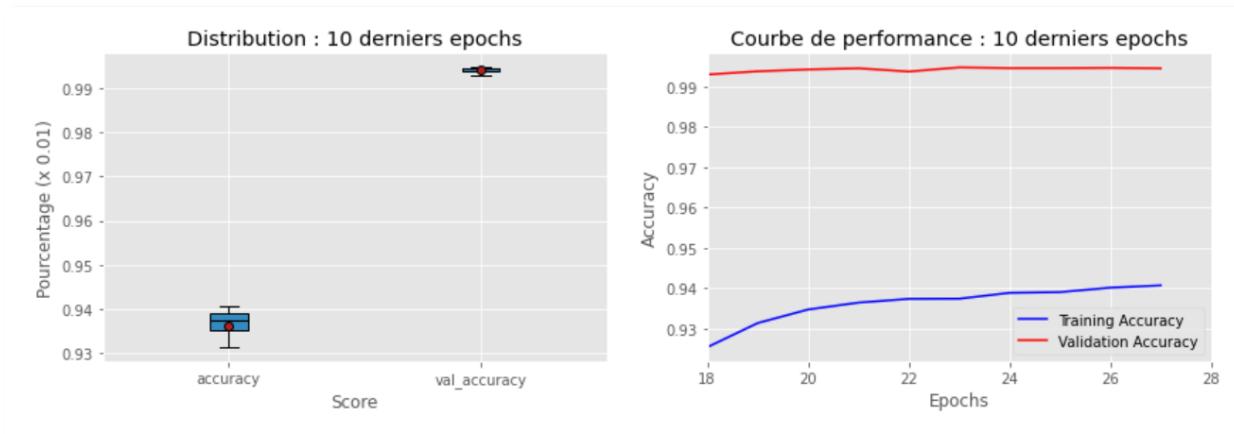


Figure 11 - VGG16 - Courbe de performance sur les dernières époques

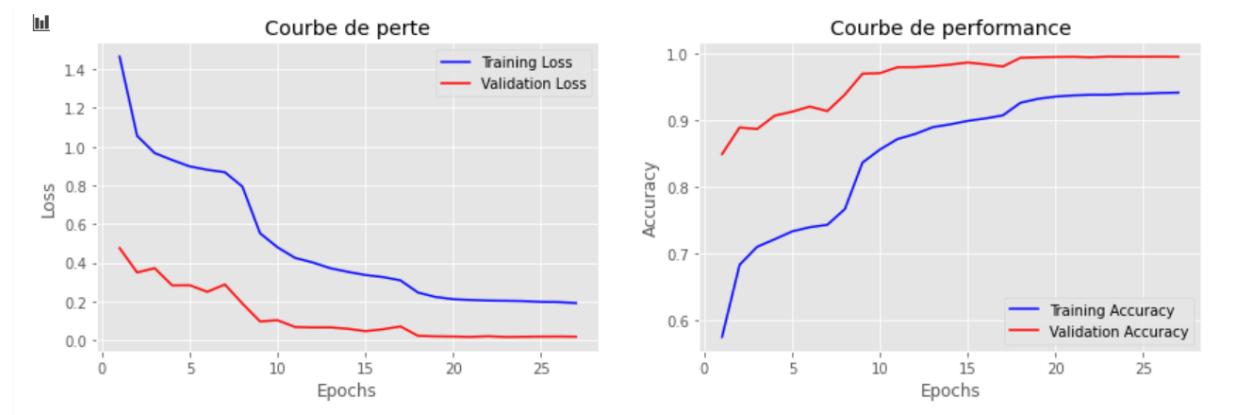


Figure 12 - VGG16 - Courbes de perte et de performance

6.1.4. Amélioration des prédictions du CNN

Au vu des résultats, le projet, pour sa partie « classification » aurait pu s'arrêter ici.

| | |
|-----------------------|------------------------|
| Fichier : | AppleCedarRust1.JPG |
| Classe prédite : | Apple_Cedar_apple_rust |
| Indice de confiance : | 100.0% |
| Fichier : | AppleCedarRust2.JPG |
| Classe prédite : | Apple_Cedar_apple_rust |
| Indice de confiance : | 100.0% |
| Fichier : | AppleCedarRust3.JPG |
| Classe prédite : | Apple_Cedar_apple_rust |
| Indice de confiance : | 100.0% |
| Fichier : | AppleCedarRust4.JPG |
| Classe prédite : | Apple_Cedar_apple_rust |
| Indice de confiance : | 100.0% |
| Fichier : | AppleScab1.JPG |
| Classe prédite : | Apple_Apple_scab |
| Indice de confiance : | 100.0% |

Figure 13 - VGG16 - Exemple de prédictions d'images de plantes

Mais même si les prédictions sur le jeu de données de test étaient à 100% correctes, nous nous sommes rendu compte que les images représentant autre chose qu'une plante était prédites comme étant une plante avec un indice de confiance important, soit très souvent une prédiction avec plus de 90% de confiance.

Quoique cela ne fût pas totalement anormal, cet effet de bord était désagréable et nous voulions résoudre ce problème.

A cet effet, une nouvelle classe nommée « Other » a été ajoutée au jeu de données.

Dans les répertoires représentant cette classe, ont été ajoutées des photos diverses d'objets ou de personnes téléchargées sur le site « [Open images dataset V6](#) ».

Le nombre de photos ajoutées a respecté la distribution du nombre d'images de plantes se trouvant dans les autres répertoires du jeu de données.

Le CNN a été entraîné sur 30 époques. Ensuite les prédictions permettaient l'identification des images de plantes et de signaler en plus, par l'ajout de cette nouvelle classe à prédire, les images qui n'étaient pas des plantes.

| | |
|-----------------------|---------------------------------------|
| Fichier : | TomatoYellowCurlVirus4.JPG |
| Classe prédictive : | Tomato__Tomato_Yellow_Leaf_Curl_Virus |
| Indice de confiance : | 100.0% |
| | |
| Fichier : | TomatoYellowCurlVirus5.JPG |
| Classe prédictive : | Tomato__Tomato_Yellow_Leaf_Curl_Virus |
| Indice de confiance : | 100.0% |
| | |
| Fichier : | TomatoYellowCurlVirus6.JPG |
| Classe prédictive : | Tomato__Tomato_Yellow_Leaf_Curl_Virus |
| Indice de confiance : | 100.0% |
| | |
| Fichier : | _chat.jpg |
| Classe prédictive : | Other |
| Indice de confiance : | 100.0% |
| | |
| Fichier : | _visage_femme.jpg |
| Classe prédictive : | Other |
| Indice de confiance : | 100.0% |

Figure 14 - VGG16 - Exemple de prédictions images de plantes et autres

6.2. Segmentation d'images

6.2.1. Premières approches

De nombreux traitements d'image sont possibles et réalisables, notamment avec la librairie OpenCV.

Lors de premiers essais, des techniques de « thresholding », ou de détection de contours sans utilisation de forme de Machine Learning ont été utilisées.

Ces méthodes qui peuvent être efficaces dans certains cas, montrent rapidement des limites.

Pour nos données, il y a par exemple des photos de feuilles de maïs prenant toute la photo. Ainsi, l'utilisation de méthode de détection de contours n'est pas efficace dans le cas de ces photos n'en présentant pas.

Le cas de photo sous ou surexposée peut aussi être compliqué à traiter par ces méthodes plus « classiques ».

En réalisant des recherches sur le sujet de la segmentation d'images, on trouve plusieurs modèles de Deep Learning permettant cette tâche.

Notamment le modèle Mask-RCNN, qui permet à la fois de détecter les objets dans l'image avec des « bounding box » et segmenter ces objets mais aussi de réaliser la classification de ces objets en de nombreuses classes.

Ainsi le choix d'utiliser des méthodes de Deep Learning semble être une option intéressante pour la tâche à réaliser.

6.2.2. Modèle de Deep Learning

6.2.2.1. Choix du modèle

Un des modèles que l'on retrouve souvent dans les problèmes de segmentation d'image est le modèle U-Net.

Ces réseaux de neurones fonctionnent avec un backbone composé de plusieurs couches de convolution et de maxpooling.

Après l'extraction de features par ce backbone, des couches de « Dé-convolution » et de « UpSampling » sont ajoutées pour permettre d'obtenir l'image de sortie.

6.2.2.2. Traitement des données

Le jeu de données utilisé (plantvillage dataset) comprend des images de feuilles des différentes classes du jeu de données : new-plant-diseases-dataset.

Cependant, dans ce cas, ces classes ne nous intéressent pas puisqu'ici nous ne sommes pas dans un cas de classification mais de segmentation d'objet dans une image.

Le but est de détecter la feuille à classifier dans l'image. Un premier essai a été réalisé en proposant au modèle des images de feuilles en entrée et les images segmentées comme label.

Cette méthode s'est avérée peu concluante, la précision du modèle n'était pas satisfaisante.

Une autre approche est de prédire des masques. Ces masques correspondent à des images de pixels prenant la valeur 1 si celui-ci correspond à l'objet à détecter et 0 sinon. On obtient visuellement des images avec un fond noir et la forme de l'objet en blanc.

Cependant pour réaliser cette prédiction, le modèle doit être entraîné avec des masques et le jeu de données n'en comporte aucun.

Ainsi il est nécessaire de transformer les images segmentées disponibles en masques.

Les lignes de codes ci-dessous permettent de faire cette transformation :

```
### Transformation des images segmentées en masques
threshold=20
for path in path_folders:
    for fichier in glob.glob(path+'/**'):
        img=cv2.imread(fichier)
        if len(img.shape)==3:

            ### La moyenne des pixels RGB est calculée et un seuil est défini pour séparer les pixels
            ### de la feuille et le reste.
            ### La valeur de 1 est attribué pour un pixel de la feuille, le pixel prend la valeur 0 sinon.

            img_mask=(img.mean(axis=2)>threshold).astype(int)
            cv2.imwrite(fichier,img_mask)
```

Figure 15 - Transformation des images segmentées en masques

Une fois les images récupérées et lues, la moyenne de chaque pixel est réalisée et si la valeur dépasse un certain seuil : « threshold », le pixel prend la valeur 1.

Ce seuil a été défini pour certains pixels noirs qui n'ont pas exactement la valeur 0. C'est notamment le cas de certains pixels au bord de l'objet.

Exemple de masque généré à partir d'une image segmentée.

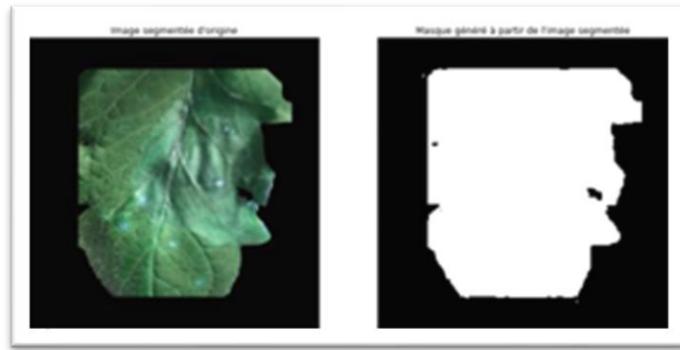


Figure 16 - Exemple de masque généré à partir d'une image segmentée

6.2.2.3. Entrainement du modèle

Avec la transformation des données, le modèle est prêt à être entrainé.

La structure du modèle est la suivante :

| Layer (type) | Output Shape | Param # | Connected to |
|--------------------------------|-----------------------|---------|-----------------------|
| <hr/> | | | |
| input_1 (InputLayer) | [None, 256, 256, 3] | 0 | |
| conv2d (Conv2D) | (None, 256, 256, 32) | 896 | input_1[0][0] |
| conv2d_1 (Conv2D) | (None, 256, 256, 32) | 9248 | conv2d[0][0] |
| max_pooling2d (MaxPooling2D) | (None, 128, 128, 32) | 0 | conv2d_1[0][0] |
| conv2d_2 (Conv2D) | (None, 128, 128, 64) | 18496 | max_pooling2d[0][0] |
| conv2d_3 (Conv2D) | (None, 128, 128, 64) | 36928 | conv2d_2[0][0] |
| max_pooling2d_1 (MaxPooling2D) | (None, 64, 64, 64) | 0 | conv2d_3[0][0] |
| conv2d_4 (Conv2D) | (None, 64, 64, 128) | 73856 | max_pooling2d_1[0][0] |
| conv2d_5 (Conv2D) | (None, 64, 64, 128) | 147584 | conv2d_4[0][0] |
| max_pooling2d_2 (MaxPooling2D) | (None, 32, 32, 128) | 0 | conv2d_5[0][0] |
| conv2d_6 (Conv2D) | (None, 32, 32, 256) | 295168 | max_pooling2d_2[0][0] |
| conv2d_7 (Conv2D) | (None, 32, 32, 256) | 590080 | conv2d_6[0][0] |
| max_pooling2d_3 (MaxPooling2D) | (None, 16, 16, 256) | 0 | conv2d_7[0][0] |
| conv2d_8 (Conv2D) | (None, 16, 16, 512) | 1180160 | max_pooling2d_3[0][0] |
| conv2d_9 (Conv2D) | (None, 16, 16, 512) | 2359808 | conv2d_8[0][0] |
| up_sampling2d (UpSampling2D) | (None, 32, 32, 512) | 0 | conv2d_9[0][0] |
| <hr/> | | | |
| concatenate (Concatenate) | (None, 32, 32, 768) | 0 | None |
| <hr/> | | | |
| conv2d_10 (Conv2D) | (None, 32, 32, 256) | 1769728 | concatenate[0][0] |
| conv2d_11 (Conv2D) | (None, 32, 32, 256) | 590080 | conv2d_10[0][0] |
| up_sampling2d_1 (UpSampling2D) | (None, 64, 64, 256) | 0 | conv2d_11[0][0] |
| concatenate_1 (Concatenate) | (None, 64, 64, 384) | 0 | up_sampling2d_1[0][0] |
| conv2d_12 (Conv2D) | (None, 64, 64, 128) | 442496 | concatenate_1[0][0] |
| conv2d_13 (Conv2D) | (None, 64, 64, 128) | 147584 | conv2d_12[0][0] |
| up_sampling2d_2 (UpSampling2D) | (None, 128, 128, 128) | 0 | conv2d_13[0][0] |
| concatenate_2 (Concatenate) | (None, 128, 128, 192) | 0 | up_sampling2d_2[0][0] |
| conv2d_14 (Conv2D) | (None, 128, 128, 64) | 110656 | concatenate_2[0][0] |
| conv2d_15 (Conv2D) | (None, 128, 128, 64) | 36928 | conv2d_14[0][0] |
| up_sampling2d_3 (UpSampling2D) | (None, 256, 256, 64) | 0 | conv2d_15[0][0] |
| concatenate_3 (Concatenate) | (None, 256, 256, 96) | 0 | up_sampling2d_3[0][0] |
| conv2d_16 (Conv2D) | (None, 256, 256, 32) | 27680 | concatenate_3[0][0] |
| conv2d_17 (Conv2D) | (None, 256, 256, 32) | 9248 | conv2d_16[0][0] |
| conv2d_18 (Conv2D) | (None, 256, 256, 1) | 33 | conv2d_17[0][0] |
| <hr/> | | | |
| Total params: 7,846,657 | | | |
| Flops: 7.64E+09 | | | |

Figure 17 - Structure du modèle U-Net

La fonction de perte choisie est utilisée couramment dans des problèmes de segmentation avec un modèle U-Net. Il s'agit du coefficient « Loss Dice ».

L'entraînement s'est fait en plusieurs étapes avec sauvegarde et chargement des poids à chaque cycle.

Le modèle a convergé très rapidement vers une précision de 95%.

6.2.3. Résultats

La meilleure appréciation des résultats est visuelle :

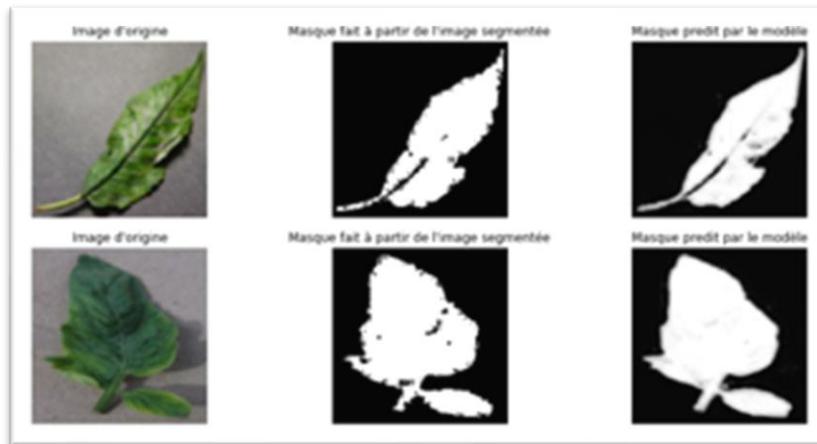


Figure 18 - Exemple de prédictions de masques du modèle, en comparaison avec des labels

La précision de 95% du modèle est à relativiser par rapport aux données d'entraînement. En effet, on peut voir que les masques utilisés pour entraîner le modèle ne sont pas parfaits.

Certains masques prédits sont plus précis que ceux générés en transformant les données. Ainsi, même un modèle réalisant des masques parfaits n'atteindrait pas 100% de précision car les masques d'entraînement ne sont pas tout à fait exacts.

Les résultats obtenus sur les feuilles des différentes classes à prédire sont cependant plus que satisfaisants. Ces masques permettent bien de détecter la feuille dans l'image.

Ce modèle de segmentation possède cependant des limites. Il ne détecte notamment pas si une feuille est bien présente ou non dans l'image. Ainsi, en lui donnant une image quelconque, le modèle va quand même essayer de prédire le masque d'une feuille dans l'image. Il serait donc nécessaire d'ajouter un neurone au modèle permettant de réaliser cette action.

6.3. Détection d'objets graphiques ou « Bounding Box »

La détection d'objet dans les images trouve de nombreux domaines d'application. Pour n'en citer que quelques-uns : analyse du trafic routier, détection des panneaux de signalisation pour la voiture autonome, détection de défauts sur une infrastructure...

L'objectif ici n'est plus de classifier une image, mais de détecter les objets au sein de celle-ci, en dessinant un rectangle entourant le plus précisément les objets recherchés et présents dans l'image.

De nombreux algorithmes permettent de réaliser cette identification d'objets graphiques : YOLO, R-CNN, Fast R-CNN, Faster R-CNN, SSD, RetinaNet...

Bien que pouvant utiliser ce type d'algorithmes performants pour identifier et encadrer chaque feuille dans l'image, nous souhaitions utiliser les résultats existants en matière de « segmentation des images » pour réaliser le « bounding box ».

En effet, la segmentation permet déjà d'obtenir le contour des feuilles dans l'image.

L'hypothèse ici a donc été de pouvoir définir une méthode qui, à partir des informations de segmentation, calcule les dimensions du cadre devant entourer chaque feuille dans l'image.

Cela éviterait une phase supplémentaire et longue d'entraînement d'un nouveau RN pour réaliser cette tâche.

Après étude de librairies python permettant de travailler sur les images, il a été implémenté une méthode qui à partir des informations de segmentation, ou plutôt à partir des masques correspondants, de calculer les zones utiles de chaque plante sur l'image pour ensuite créer le cadre graphique d'entourage de l'objet identifié.

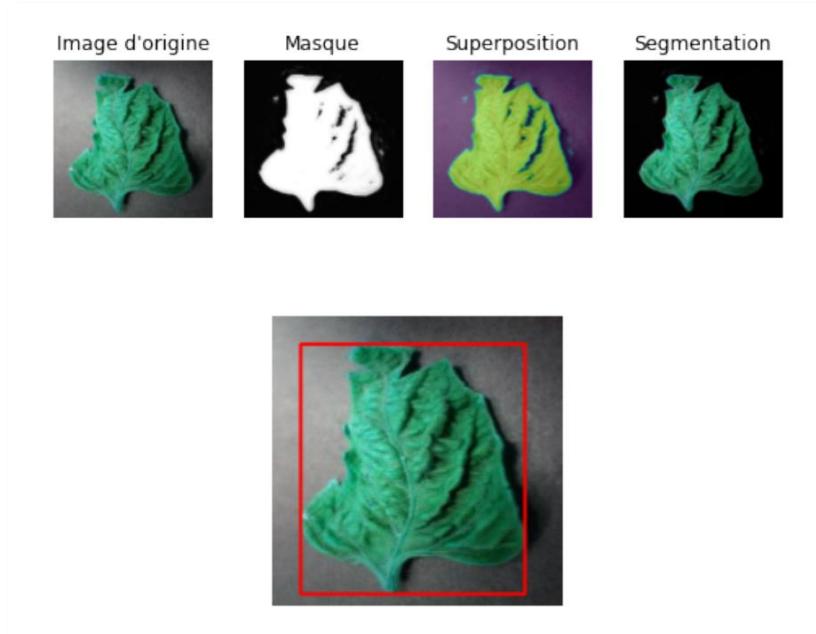


Figure 19 - Détection d'objet - Bounding box

La limite à cette méthode a été la qualité des masques issus de la segmentation. En effet, pour calculer précisément les dimensions du cadre d'entourage d'une plante, il est nécessaire d'avoir des contours nets de celle-ci.

Les résultats obtenus sont conformes aux attentes initiales et donc l'entraînement du RN supplémentaire n'a pas été nécessaire.

7. Description des travaux réalisés

7.1. Répartition de l'effort sur la durée et dans l'équipe

Le projet a été segmenté en un ensemble de tâches :

| Tâche | Qui | Quand |
|------------------------------------------------------|-----------------|------------|
| Sélection du projet | Equipe | 2020-S43 |
| Point Projet (réunion de lancement) | Equipe + Client | 27/10/2020 |
| Cadrage du projet (périmètre, objectifs, ressources) | Equipe | 2020-S44 |
| Inventaire des données | Noël MAURICE | 2020-S45 |
| Analyse des données | Loïc AXILAIIS | 2020-S45 |
| Documentation de l'exploration des données | Mathis RAVAILAO | 2020-S45 |
| Point Projet | Equipe + Client | 05/11/2020 |
| Structuration de l'architecture des dossiers | Noël MAURICE | 2020-S46 |
| Visualisation des données | Loïc AXILAIIS | 2020-S46 |
| Complément à la visualisation des données | Mathis RAVAILAO | 2020-S46 |
| Point Projet | Equipe + Client | 10/11/2020 |
| Expérimentation sur les modèles de classification | Noël MAURICE | 2020-S47 |
| Expérimentation sur PCA et Clustering | Mathis RAVAILAO | 2020-S47 |
| Expérimentation sur Streamlit | Loïc AXILAIIS | 2020-S47 |
| Sélection d'un modèle d'identification | Equipe | 2020-S47 |
| Point Projet | Equipe + Client | 17/11/2020 |
| Entraînement du CNN d'identification V1 | Noël MAURICE | 2020-S48 |
| Réalisation du site Streamlit V1 | Loïc AXILAIIS | 2020-S48 |
| Point Projet | Equipe + Client | 26/11/2020 |
| Migration sur Collab | Noël MAURICE | 2020-S49 |
| Implémentation du CNN d'identification V2 | Noël MAURICE | 2020-S49 |
| Réalisation du site Streamlit V2 (multipages) | Loïc AXILAIIS | 2020-S49 |
| Optimisation du CNN d'identification V2 | Noël MAURICE | 2020-S50 |
| Optimisation du site Streamlit multipages | Loïc AXILAIIS | 2020-S50 |
| Modèle de segmentation | Mathis RAVAILAO | 2020-S50 |
| Point Projet | Equipe + Client | 09/12/2020 |
| Implémentation des bounded box | Noël MAURICE | 2020-S51 |
| Finalisation du site Streamlit multipages | Loïc AXILAIIS | 2020-S51 |
| Point Projet | Equipe + Client | 15/12/2020 |
| Rédaction du rapport technique d'évaluation | Equipe | 2020-S52 |
| Soutenance du projet | Equipe + Client | 07/01/2021 |

Le diagramme de Gantt reprenant cette distribution des tâches est le suivant :

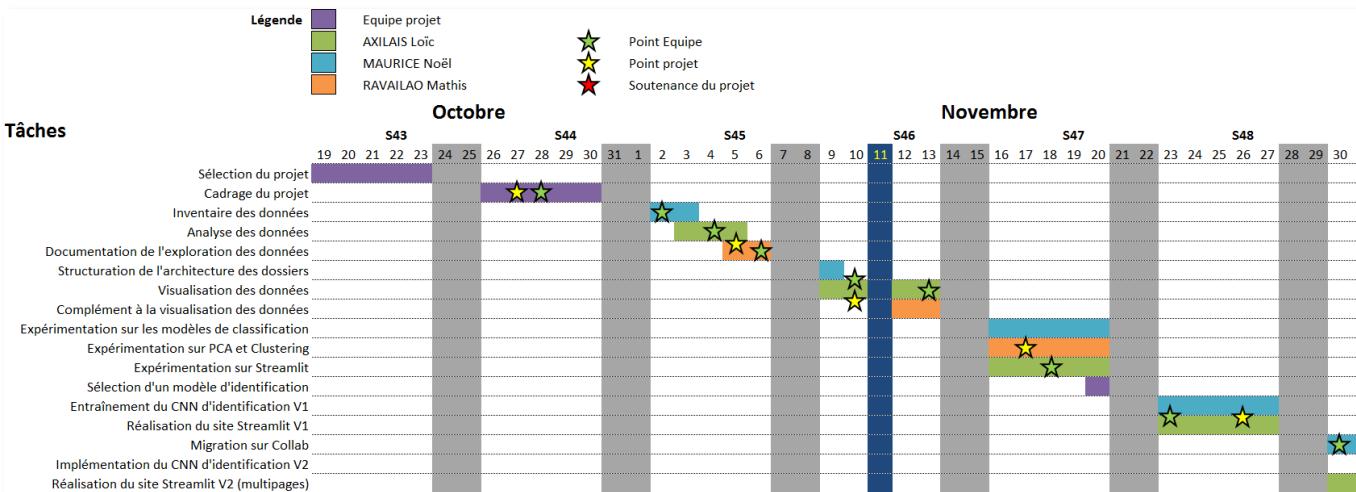


Figure 20 - Diagramme de Gantt - Partie 1

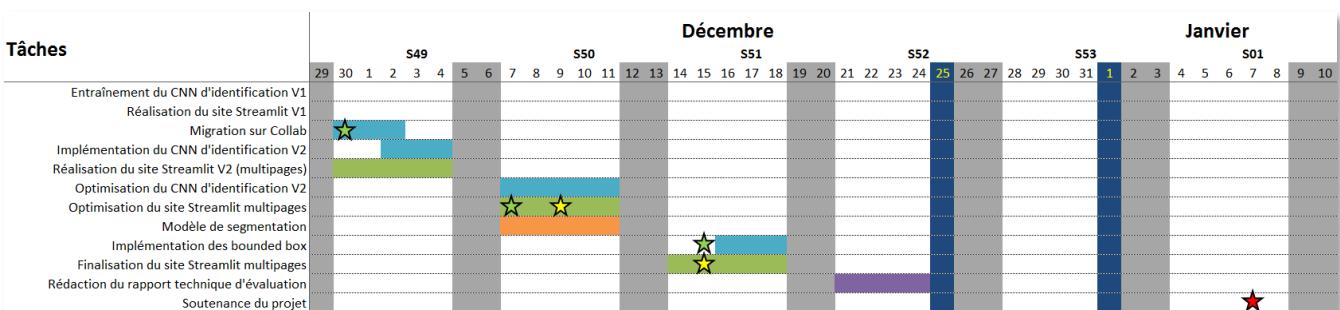


Figure 21 - Diagramme de Gantt - Partie 2

Remarque : Même si chaque membre de l'équipe a réalisé des sujets de développement spécifique, nous avons pratiqué une relecture croisée du code et de la documentation pour nous assurer de la qualité des livrables.

7.2. Bibliographie

Sans que cette liste soit exhaustive, voici énumérées des ressources ou documentations directement exploitables dans le cadre du projet :

7.2.1. Scientifique

<https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=8632907>

RIC-Unet: An improved neural network based on Unet for nuclei segmentation in histology images

<https://www.sciencedirect.com/science/article/abs/pii/S0168169918315308>

Analysis of transfer learning for deep neural network based plant classification models

<https://hal.archives-ouvertes.fr/hal-01629183/document>

Plant identification based on noisy web data: the amazing performance of deep learning (LifeCLEF 2017)

<https://arxiv.org/pdf/1603.07285.pdf>

A guide to convolution arithmetic for deep learning

7.2.2. Technique

<https://www.kaggle.com/vipooooool/new-plant-diseases-dataset>

Données pour la classification des images de plante.

<https://www.kaggle.com/abdallahalidev/plantvillage-dataset/version/1>

Données pour la segmentation et le calcul des « bounding boxes » des images.

<https://storage.googleapis.com/openimages/web/download.html>

Données complémentaires pour la différentiation des photos de plantes ce celles qui ne sont pas des photos de plantes.

<https://machinelearnia.com/>

<https://www.youtube.com/c/MachineLearnia/playlists>

Site « MachineLearnia » dédié à l'IA et playlists composées de très nombreux tutoriels de qualité dans le domaine de l'IA – Réalisations de Guillaume Saint-Cirgue.

<https://www.udemy.com/course/introduction-au-machine-learning/>

<https://www.udemy.com/course/intelligence-artificielle-pour-le-business/>

<https://www.udemy.com/course/le-deep-learning-de-a-a-z/learn/>

<https://www.udemy.com/course/intelligence-artificielle-az/>

Formations vidéo - Hadelin de Ponteves, AI Entrepreneur sur le Machine Learning et le Deep Learning.

<https://www.youtube.com/channel/UCn09iU3hS5Fpxv0XniGv2FQ/playlists>

Tutoriels sur le Deep Learning et notamment sur la segmentation d'images et le bounding box.

https://play.google.com/store/apps/details?id=org.plantnet&hl=en_US

<https://www.plantsnap.com/>

Applications d'identification de plantes

7.3. Difficultés rencontrées lors du projet

7.3.1. Timing Projet - Formation

La littérature scientifique du Net au sujet des réseaux de neurones à convolution (CNN) est abondante.

Encore faut-il savoir quelles informations exploiter pour pouvoir rechercher les rapports, thèses ou documentations sur celles-ci, et mettre en œuvre les algorithmes ou méthodologies correspondants.

Dès le démarrage du projet un écart temporel entre les besoins de connaissance et la formation de « DataScientest » est apparu.

En effet, les savoirs à propos des CNN apparaissent près de la fin de cette formation. Cela est en soit compréhensible, tant les prérequis sont importants pour pouvoir réaliser de la classification d'images à l'aide des CNN.

Néanmoins, on ne pouvait pas attendre la fin de la formation pour entreprendre le projet. Donc en plus du rythme soutenu des cours, il a fallu se former en parallèle aux concepts nécessaires au projet.

Certes cette formation en marge n'était pas optimale et bon nombre de techniques n'ont pu être mises en œuvre qu'en cours ou fin de projet.

Pour n'en citer que quelques-unes : le « transfer learning », les nombreux callbacks, la personnalisation des RN et de leurs couches, etc.

Cela explique en partie l'approche expérimentale du projet et de son évolution sous forme itérative.

7.3.2. Ressources techniques

L'entraînement de réseaux de neurones demande des ressources système importantes. Bien que la première itération du projet à travers un CNN classique ait été à peu près possible sur un système local, l'entraînement de CNN plus évolués a vite demandé la puissance de serveurs évolués.

« DataScientest » a mis à notre disposition les ressources d'un serveur mais celui-ci s'est montré très en deçà de la puissance attendue pour continuer notre projet.

La plateforme « Google colab » offrait ce type de service mais de manière limitée. En effet, le temps d'exécution d'un algorithme n'est pas garanti et une session peut s'arrêter n'importe quand, de plus les ressources système qu'offre cette plateforme sont mutualisées ce qui ne permet pas de savoir pendant combien de temps on peut en disposer.

Donc pour pouvoir utiliser cette plateforme, il a fallu réadapter notre code source pour permettre l'exploitation des jeux de données de plusieurs giga-octets, puis de permettre l'enregistrement des données d'entraînement en continu car lorsqu'une session de cette plateforme s'arrête, tout est perdu.

Pour en revenir au point précédent sur le timing du projet par rapport à la formation, nous n'avions alors pas suivi complètement le cours sur les RN et leur personnalisation. Cela aurait grandement facilité la tâche du moment et résolu en partie ces problèmes de ressources limitées dans le temps. Par exemple en utilisant les fonctions callbacks pour l'enregistrement systématique des données d'entraînement, ou bien pour que le CNN gère automatiquement l'hyper-paramètre « Learning rate », pour ne citer que ces points.

7.3.3. Echange d'information dans le code et protocoles fonctionnels

Nous nous sommes répartis les tâches de façon à équilibrer la charge. Chacun œuvrant dans son propre environnement de travail, il n'a pas toujours été simple pour chacun d'intégrer le développement des autres dans son code.

Voici quelques problèmes rencontrés :

7.3.3.1. Protocole d'échange entre systèmes

Même si nous avons rapidement acté une structure de répertoires pour élaborer nos développements, certains chemins d'accès ont pu diverger, notamment, au début du projet rendant impossible l'exécution des traitements.

Une harmonisation a donc été décidée sur les noms des répertoires et des fichiers. Seuls les mêmes noms de variables pour les répertoires et les fichiers étaient utilisés dans le code. Les valeurs de ces variables étaient définies dans un fichier propre à chacun et en fonction de sa configuration propre.

En fait ce fichier jouait le rôle d'interface entre le système hôte et la structure interne du programme.

7.3.3.2. Intégration des fonctions inter-fichier

Pour extraire une portion de code d'un notebook et l'intégrer dans un autre code, il est souvent nécessaire de faire quelques ajustements. Ces ajustements doivent être repris systématiquement de manière « manuelle » lorsque la version du notebook d'origine évolue.

En fait, une programmation orienté objet avec ses mécanismes d'héritage, de polymorphisme, de surcharge, etc. aurait été très profitable ici mais le temps manquait pour sa mise en œuvre.

7.3.4. Environnements hétérogènes

Il existe des différences de comportement des traitements selon qu'on les exécute sous Windows ou Linux (ex : intolérance de Windows sur les chemins d'accès aux fichiers de plus de 260 caractères).

Il n'y a pas d'harmonisation des signatures d'appel aux fonctions entre les différentes librairies python ce qui oblige parfois à jongler avec des conversions de formats entre bibliothèques (par exemple : les formats vidéo attendus sont différents entre le framework streamlit, les différentes librairies python comme cv2, PIL et tensorflow).

Lors du démarrage du projet, nous étions en mode exploration, expérimentation, chacun sur ses propres problématiques. Nous n'avons donc pas engagé une démarche de gestion de configuration « requirement » sur les bibliothèques python utilisées dans le projet.

Bien entendu, dans le cas d'une mise en production, une gestion de l'environnement de développement aurait été nécessaire pour garantir que l'application soit 100% compatible avec les systèmes « clients », ce que nous ne garantissons pas à ce jour.

8. Succès du projet et évolutions possibles

8.1. Objectifs initiaux

Rappel de la demande du client :

L'objectif de ce projet est de localiser et classifier l'espèce d'une plante dans une image.

Une fois la classification faite, vous pouvez retourner à l'utilisateur une description de celle-ci et identifier les possibles maladies. L'application sera donc capable à partir d'une image prise par l'appareil photo de donner une succession d'informations à l'utilisateur.

8.2. Facteurs de validation

Rappel des conditions de validation du projet définies par le client :

- *Exploration des données (vision d'ensemble sur les données)*
- *Visualisations commentées et analysées*
- *Rapport technique + programmes + Github*
- *Une démo (Streamlit)*

Partie optionnelle du projet :

- Identifier que l'image soumise ne représente pas une feuille de plante,
- Segmentation de l'image,
- Ajout de « bounding box » sur l'image d'origine.

8.3. Résultats du projet

Les objectifs ont été atteints à travers les fonctionnalités suivantes :

8.3.1. Traiter une photographie soumise par l'utilisateur

L'application permet à l'utilisateur d'utiliser une photographie dans une bibliothèque prédéfinie d'images de plantes, ou d'importer une image de son choix à partir de son système informatique.

8.3.2. Identifier une plante dans une image

Le modèle de classification reconnaît avec une précision moyenne de plus de 99% les plantes issues du jeu de données, soit les plantes suivantes : Cerise et griotte, Courge, Fraise, Framboise, Maïs, Myrtille, Orange, Pêche, Poivron, Pomme, Pomme de terre, Raisin, Soja, Tomate.

Le système est aussi capable d'indiquer à l'utilisateur si la photographie représente bien une feuille de plante ou non car il a aussi été entraîné sur des objets graphiques ne représentant pas des plantes.

8.3.3. Identifier les maladies possibles

Avec la même performance que précédemment, notre modèle de classification est capable de reconnaître avec une précision moyenne de plus de 99% une des maladies suivantes : Alternariose, Bactériose, Brûlure, Cercosporiose, Cladosporiose, Corynesporiose, Helminthosporiose du nord, Isariopsis, Maladie du bois, Mildiou, Oïdium, Pourriture noire, Rouille commune, Rouille de virginie, Septoriose, Tavelure, Tétranyque, Verdissement, Virus des feuilles jaunes en cuillère, Virus Mosaïque.

8.3.4. Retourner une description à l'utilisateur

Une fois l'identification de la maladie réalisée, notre système collecte des informations relatives à la plante identifiée à partir du site internet [PlantVillage](#).

Ce site est spécialisé sur les plantes et leurs maladies et sert donc de référence pour les informations relatives aux plantes identifiées par le système..

8.3.5. Extraire la plante de l'image

La partie « segmentation » de l'application est capable d'isoler la feuille d'une plante dans l'image en la présentant sur un fond noir avec une performance proche de 95%.

Ensuite, à partir de cette étape, la feuille de la plante extraite de l'image en calculant ses limites extérieures. Ces limites sont représentées par une « bounding box », ou un cadre entourant, affiché autour de la feuille.

8.3.6. Valorisation des résultats

8.3.6.1. Rapport Technique d'Evaluation

Le livrable « Rapport Technique d'Evaluation » est composé de ce document et de ses annexes.

Ce document présente de manière la plus exhaustive possible le déroulement complet du projet.

8.3.6.2. Interface graphique

L'interface graphique de démonstration des résultats du projet, et sur les préconisations du client, a été réalisée avec le framework python « Streamlit ».

Cette interface est un site internet comprenant les pages « web » suivantes :

- Présentation : Cette page présente le projet pyStill ;
- Jeux de données : Cette page recense les jeux de données mis à notre disposition ;
- Exploration : Cette page présente les explorations menées sur les jeux de données avec des visualisations commentées et analysées ;
- PCA : Cette page présente nos travaux pour la réduction des dimensions sur les images ;
- Méthodologie : Cette page présente notre approche du projet et la façon dont nous avons exploité les sources de données ;
- Modélisation : Cette page présente nos différents modèles et leur évolution à travers le projet ;
- Segmentation : Cette page est une démonstration des résultats du modèle de segmentation et de l'extraction d'objet avec « bounding box » ;
- Identification : Cette page est une démonstration des résultats du modèle de classification ;
- Conclusion : Cette page présente la conclusion du projet pyStill.

8.3.6.3. Plateforme de contrôle de gestion Github

Les développements relatifs au projet (notebooks, librairies, site Streamlit) et notre documentation (dont ce rapport) se trouvent dans le repository Github : <https://github.com/DataScientest/Pystill>

8.4. Perspectives d'évolutions possibles

Suite à ce qui précède, on peut mesurer les facteurs de réussite du projet et ceux-ci sont très positifs.

Ce projet s'inscrit dans le cursus de formation et des évolutions pourraient être envisagées pour évoluer vers une application commerciale, où tout du moins plus utile au grand public :

8.4.1. Robustesse du modèle

Les prédictions sur les plantes présentes dans le jeu de données sont excellentes

Mais au cours de ce projet, nous avons constaté un effet de bord pour la non-détection d'images hors périmètre du projet et cela a été expliqué précédemment dans ce document.

Maintenant et pour la suite à donner à ce projet, il serait peut-être bien de rendre le CNN de classification plus robuste aux effets de bord.

En effet, l'algorithme analyse un nombre limité de plantes. Nous avons alors constaté que lorsqu'il reçoit une image d'une plante pour laquelle il n'a pas été entraîné, il identifie cette plante comme étant connue.

Certes, l'indice de confiance de la prédiction est, dans ce cas, plus faible qu'avec une plante connue mais cet indice reste tout de même assez important.

8.4.2. Domaine de prédiction

Nos modèles de prédiction présentent de bonnes performances mais leur faiblesse principale est le nombre de classes prédites (38 classes de couple plante-maladie et une classe « autres »).

Pour une application professionnelle utilisable dans le monde agricole, il faudrait augmenter le nombre de classes prédites de manière significative.

Rien qu'en France, il existe une centaine de légumes cultivés. L'Organisation des Nations Unies pour l'Alimentation et l'Agriculture (FAO) estime que, sur près de 250 000 variétés végétales propres à la culture, on n'en cultive aujourd'hui qu'environ 7 000.

Chaque plante est associée à un nombre important de maladies. Les maladies peuvent être dues à des virus, des bactéries ou des champignons. Les champignons causent à eux seuls plus de 100 000 maladies.

Ces quelques chiffres montrent bien que le chemin est encore long pour couvrir des besoins plus professionnels même si ces chiffres délimitent le maximum du domaine de prédiction.

8.4.3. Gamme de services

L'identification et la classification des plantes et de leurs maladies respectives est une étape nécessaire mais insuffisante pour répondre aux enjeux culturaux.

Savoir qu'une feuille est malade ne donne pas d'information sur l'état de santé générale de la plante. Il faudrait donc un système capable de quantifier « l'ampleur des dégâts » dans un premier temps.

Puis dans un second temps, le système devrait être capable de proposer un traitement curatif ou préventif et un dosage de produits phytosanitaires pour endiguer la maladie.

Enfin pour être complet, le système devrait être capable de donner des indications sur le coût de traitement à l'hectare (coût du travail, coût des traitements, nombre de passages, évaluation des pertes en cas d'inaction, ...) permettant ainsi aux producteurs de vérifier la rentabilité économique du traitement et de mesurer éventuellement son impact écologique.

9. Bilan & Conclusion

9.1. Bilan scientifique

Ce projet s'est appuyé principalement sur les concepts scientifiques pour l'identification et la classification d'images et de leur contenu.

Sa réalisation a été itérative, s'incluant dans une démarche d'expérimentation scientifique : « Hypothèse – Expérimentation - Validation ou non ».

Le résultat a été la validation des différentes phases du projet, la validation des méthodes utilisées pour la classification des plantes et de leurs maladies, et des méthodes de détection et d'isolation des plantes au sein des images.

Le client a approuvé les résultats. Les mesures de performances, les tests de classification et d'identification ont confirmé ces résultats très satisfaisants.

9.2. Bilan technique et professionnel

Par ailleurs et à travers ce projet de deep learning, nous avons pu mettre en pratique les concepts, méthodes et technique acquises lors de la formation de « Data Scientist » dispensée par « Datascientest ».

Notre démarche en mode « client – fournisseur » nous a permis de présenter une solution fonctionnelle à une problématique issue du monde professionnel.

9.3. Bilan de l'équipe

9.3.1. Investissement

Chaque membre de l'équipe s'est largement mobilisé pour atteindre les objectifs et a fait preuve d'une forte motivation.

La communication a été privilégiée avec des points hebdomadaires entre les membres du groupe : points permettant notamment la revue du code, l'avancement des travaux, la proposition de solutions, la validation du projet, etc.

9.3.2. Timing

Même si le timing entre le démarrage et l'avancement du projet et les cours nécessaires pour y arriver n'a vraiment pas été synchrone, notre travail de recherche et de documentation nous a permis le respect des délais de réalisation à chaque étape du projet, et même parfois avec une marge temporelle positive pour les jalons définis par le client « Datascientest ».

9.3.3. Homogénéité du groupe et complémentarité des compétences

Nous n'avons pas eu à souffrir d'écart majeur de niveau technique sur le développement en python des différents membres de l'équipe.

La complémentarité des compétences et savoirs des membres de l'équipe a permis la bonne réalisation du projet sans soucis majeurs.

Nous avons même pu nous répartir les tâches de façon équitable et selon les aspirations de chacun.

9.4. Conclusion

Cette formation, ce projet, les expérimentations réalisées, les solutions trouvées sont autant de facteurs qui nous confortent pour notre avenir professionnel dans le monde de la « data science ».

Tout cela ajoute des compétences nouvelles et à haute valeur ajoutée à nos cursus professionnels respectifs : n'était-ce pas là l'objectif à atteindre ?

En tout cas, nous avons vraiment apprécié cette expérience qui n'est que précurseur à d'autres projets dans la « data science ».

10. Annexes

10.1. Dataviz des jeux de données

dataviz

December 25, 2020

1 Plant expert at your finger

1.1 Première exploration

1.1.1 Inventaire des jeux de données

Afin de réaliser un inventaire des informations disponibles, nous avons téléchargé les jeux de données et parcourus les fichiers.

Les jeux de données **cocodataset** et **googleapis** ont été ignorés car ils contiennent énormément d'images ne concernant pas les plantes.

!/ Conditions d'utilisation de ce notebook :

* Télécharger les 5 jeux de données compressés suivants :
 * plantclef-2019-amazon-rainforest-plants-images * v2-plant-seedlings-dataset * new-plant-diseases-dataset * plant-disease * plantvillage-dataset * Décompresser chaque jeu de données dans un répertoire portant son nom * Déposer les 5 répertoires dans le même répertoire commun et affecter ce répertoire racine à la variable dir_data du fichier ressources.py

```
[1]: # Initialisation des constantes
import os
import sys
sys.path.append(os.path.abspath(os.path.join(os.getcwd(), os.pardir)))
from lib import ressources
racine = os.path.abspath(os.path.realpath(ressources.dir_data))
sep = ressources.csv_separator
ossep = os.sep
```

Nous récupérons la liste des fichiers disponibles dans le répertoire de dépôt des jeux de données (répertoire 'racine').

Les chemins d'accès aux fichiers sont stockés dans le fichier filelist.txt en vue d'une réutilisation ultérieure

```
[7]: # Récupération de la liste des fichiers de données
from lib.tools import listdirectory
filelist = listdirectory(racine)
fichier = open(racine + ossep + 'filelist.txt', "w")
try:
    fichier.write('\n'.join(filelist))
finally:
    fichier.close()
```

Nous récupérons la liste des fichiers précédemment stockée dans le fichier filelist.txt
 Cette étape n'a d'intérêt que lorsque l'on souhaite exécuter ce notebook en plusieurs étapes sans avoir à rescanner le répertoire 'racine'.

```
[22]: fichier = open(racine + ossep + 'filelist.txt', "r")
filelist = fichier.readlines()
filelist = [x.strip('\n') for x in filelist]
```

1.1.2 Collecte de métadonnées

Afin de récupérer quelques informations sur les fichiers, nous optons pour un traitement en multi-processing (sur les conseils de Thomas).

L'objectif est de paralléliser la collecte sur plusieurs fichiers en même temps pour réduire la durée de traitement.

D'abord, interrosons nous au nombre de coeurs disponibles sur la machine.

Nous garderons un cœur libre pour pouvoir travailler sur d'autres sujets en parallèle.

Nous utiliserons donc (Nombre de CPU disponibles - 1) pour le multiprocessing.

```
[10]: cpuCount = os.cpu_count()
print('Nombre de CPU disponibles :', cpuCount)
nprocs = cpuCount - 1
print('Nombre de CPU utilisés pour le multiprocessing :', nprocs)
```

```
Nombre de CPU disponibles : 8
Nombre de CPU utilisés pour le multiprocessing : 7
```

Nous instancions un pool de process et appellons la méthode get_info sur chaque fichier de l'inventaire.

Les résultats de la collecte sont sauvegardés dans le fichier filelist.csv

Nous avons été confronté à un problème d'exécution avec Jupyter sous Windows. L'astuce (non détaillée ici) consiste à déplacer la méthode get_info dans un module à part et à instancier le pool sous la condition `__name__ == '__main__'`

```
[23]: import multiprocessing as mp
from lib.tools import get_info # Astuce pour Jupyter + Windows
from itertools import repeat

if __name__ == '__main__': # Astuce pour Jupyter + Windows
    # Collecter les métadonnées
    pool = mp.Pool(nprocs)
    results = pool.starmap(get_info, zip(filelist, repeat(sep), ↴
                                         repeat(racine)), 4)
    pool.close()
    pool.join()
    # Stocker les métadonnées dans un fichier exploitable ultérieurement
    fichier = open(racine + ossep + 'filelist.csv', "w")
    try:
        head = 'dataset' + sep + 'folder' + sep + 'file' + sep + 'ext' + sep + ↴
               'size' + sep + 'height' + sep + 'width' + sep + 'channel' + sep + 'color'
        fichier.write(head)
```

```

        fichier.write(head + '\n')
        fichier.write('\n'.join(results))
    finally:
        fichier.close()

```

1.1.3 Valorisation des métadonnées

Les informations collectées à l'étape précédente nécessitent quelques aménagements (enrichissement, harmonisation, ...).

Le fichier filelist.csv est retraité à l'aide du module pandas pour en simplifier la manipulation.

```
[24]: import pandas as pd
df = pd.read_csv(racine + ossep + 'filelist.csv', sep=sep, encoding='ansi')
# Mettre l'extension du nom du fichier en majuscule
df['ext'] = df['ext'].apply(lambda x: x.upper())
# Harmoniser les noms d'extension
df = df.replace({'ext': {'JPEG': 'JPG'}})
# Calculer le nombre de fichiers par type d'extension et par groupe
df.groupby(['dataset', 'ext']).agg({'file':'count'})
```

| | | file |
|------------------------------------------------|---------|------------|
| | dataset | ext |
| new-plant-diseases-dataset | | JPG 87900 |
| plant-disease | | JPG 108608 |
| | | PNG 2 |
| plantclef-2019-amazon-rainforest-plants-images | | CSV 1 |
| | | JPG 3142 |
| | | XML 3133 |
| plantvillage-dataset | | JPG 162914 |
| | | PNG 2 |
| v2-plant-seedlings-dataset | | PNG 11078 |

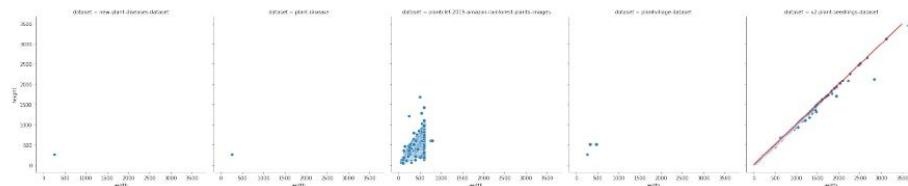
Nous constatons que le jeu de données **plantclef-2019-amazon-rainforest-plants-images** contient d'autres fichiers que des images. Nous disposons d'un fichier CSV et d'un ensemble de fichiers XML :

* Le fichier CSV synthétise la classification d'un peu plus de 448 000 photos mais la base d'images récupérées n'en contient que 3142, ce n'est donc qu'un petit sous-ensemble. * Les fichiers XML sont des fichiers compagnons des fichiers JPG et décrivent la classification de la plante. Ils contiennent des informations que l'on peut retrouver à l'identique dans le fichier CSV.

Etudions les images d'un peu plus prêt et notamment la répartition de leur taille :

```
[25]: # Calculer la répartition des tailles d'image (hauteur par largeur en fonction
      # du dataset et de la couleur)
df_dim = df[df.ext.isin(['JPG', 'PNG']) & ~df[['size', 'height', 'width']].
      #isna().any(axis = 1)].groupby(['dataset', 'color', 'height', 'width'],
      #as_index=False).agg({'file':'count'})
# Afficher la répartition des tailles d'image
```

```
import seaborn as sns
import matplotlib.pyplot as plt
sns.relplot(x='width', y='height', col='dataset', data=df_dim)
plt.plot([0,3500],[0,3500], 'r');
```



Quelques précisions :

- * **new-plant-diseases-dataset** : - 87 900 photos en couleur de dimension 256 x 256 au format JPG - Toutes les photos sont uniques
- * **plant-disease** : - 108 610 photos en couleur de dimension 256 x 256 au format JPG - Toutes les photos sont en doublons (il n'y a en fait que 54305 photos) - 2 photos (en doublon) en couleur de dimension 256 x 256 au format PNG
- * **plantvillage-dataset** : - 108 610 photos en couleur de dimension 256 x 256 au format JPG : - 54305 photos normales - 54305 photos ségmentées (sur fond noir) - 54306 photos en noir et blanc de dimension 256 x 256 au format JPG - Ce sont les mêmes photos que celle de plant-disease
- * **plantclef-2019-amazon-rainforest-plants-images** : - La base contient 3040 photos en couleur, 59 photos en noir et blanc et 43 photos corrompues - Toutes les images ne sont pas de même dimension
- * **v2-plant-seedlings-dataset** : - 548 photos ne peuvent être lues par OpenCV - Les photos ne sont pas toutes de même dimension mais elle sont majoritairement carrés

1.2 Conclusions :

Les images concernant les maladies (new-plant-diseases-dataset, plant-disease et plant-village-dataset) sont les plus prometteuses en terme de volume mais aussi d'organisation car elles possèdent les caractéristiques suivantes :

- * Elles ont toutes la même dimension
- * Elles sont déjà classées (la classe est définie par le nom du dossier les contenant)
- * Elles sont déjà organisées en jeu d'entraînement, de validation et de test
- * Elles sont structurées au format KERAS
- * Un sous-ensemble d'images existe en couleur, en échelle de gris et en segmenté

1.3 Deuxième exploration

1.3.1 Classification

Nous poussons l'analyse un peu plus loin en nous concentrant sur les images concernant les maladies

```
[59]: df_disease = df[df['dataset'].isin(['new-plant-diseases-dataset',  
                                     'plant-disease', 'plantvillage-dataset'])]  
df_disease.height = df_disease.height.astype(int)  
df_disease.width = df_disease.width.astype(int)  
df_disease.channel = df_disease.channel.astype(int)  
# Suppression des doublons dans le jeu de données plant-disease
```

```

df_disease['doublon'] = df_disease[df_disease['dataset']=='plant-disease'].
    ↪folder.apply(lambda x: True if x.split(ossep)[1]=='dataset' else False)
df_disease = df_disease.
    ↪drop(df_disease[(df_disease['dataset']=='plant-disease') &_
        ↪(df_disease['doublon'])].index)
df_disease.drop('doublon', axis=1, inplace=True)

```

Nous récupérons la classe de la plante, de la maladie et du type d'échantillon grâce au nom du répertoire dans lequel se trouve l'image.

```

[60]: def get_last_segment(val, pos):
    seg = val.split(ossep)
    last_seg = seg[len(seg)-1]
    if '___' in last_seg:
        return last_seg.split('___')[pos]
    else:
        return ''

def get_sample_type(val):
    seg = val.split(ossep)
    sample_type = ['train', 'valid', 'test', 'color', 'grayscale', 'segmented']
    for st in sample_type:
        if st in seg:
            return st
    return ''

df_disease['plant'] = df_disease.folder.apply(lambda x: get_last_segment(x, 0))
df_disease['disease'] = df_disease.folder.apply(lambda x: get_last_segment(x, ↪
    ↪1))
df_disease['sample'] = df_disease.folder.apply(lambda x: get_sample_type(x))

```

Nous traduisons les classes de plantes et des maladies

```

[61]: plant_translation = {
    'Apple' : 'Pomme',
    'Blueberry' : 'Myrtille',
    'Cherry_(including_sour)' : 'Cerise et',
    ↪griotte',
    'Corn_(maize)' : 'Maïs',
    'Grape' : 'Raisin',
    'Orange' : 'Orange',
    'Peach' : 'Pêche',
    'Pepper,_bell' : 'Poivron',
    'Potato' : 'Pomme de',
    ↪terre',
    'Raspberry' : 'Framboise',
    'Soybean' : 'Soja',
    'Squash' : 'Courge',
    'Strawberry' : 'Fraise',
    'Tomato' : 'Tomate'}

```

```

disease_translation = { 'Apple_scab' : 'Tavelure',
                        'Bacterial_spot' : 'Bactériose',
                        'Black_rot' : 'Pourriture',
                        'noire',
                        'Cedar_apple_rust' : 'Rouille de',
                        'virginie',
                        'Cercospora_leaf_spot_Gray_leaf_spot' : '',
                        'Cercosporiose',
                        'Common_rust_' : 'Rouille',
                        'commune',
                        'Early_blight' : '',
                        'Alternariose',
                        'Esca_(Black_Measles)' : 'Maladie du',
                        'bois',
                        'Haunglongbing_(Citrus_greening)' : '',
                        'Verdissement',
                        'Late_blight' : 'Mildiou',
                        'Leaf_Mold' : '',
                        'Cladosporiose',
                        'Leaf_blight_(Isariopsis_Leaf_Spot)' : 'Isariopsis',
                        'Leaf_scorch' : 'Brulure',
                        'Northern_Leaf_Blight' : '',
                        'Helmintosporiose du nord',
                        'Powdery_mildew' : 'Oïdium',
                        'Septoria_leaf_spot' : 'Septoriose',
                        'Spider_mites Two-spotted_spider_mite' : 'Tétranyque',
                        'Target_Spot' : '',
                        'Corynesporiose',
                        'Tomato_Yellow_Leaf_Curl_Virus' : 'Virus des',
                        'feuilles jaunes en cuillère',
                        'Tomato_mosaic_virus' : 'Virus',
                        'Mosaique',
                        'healthy' : 'Plante',
                        'saine'},

df_disease['plant_fr'] = df_disease.plant.apply(lambda x: plant_translation[x] if x != '' else '')
df_disease['disease_fr'] = df_disease.disease.apply(lambda x:disease_translation[x] if x != '' else '')
# Sauvegarde de la liste des fichiers concernant les maladies dans le fichier
# disease.csv pour une réutilisation ultérieure
df_disease.to_csv(racine + ossep + 'disease.csv', sep=sep, index=False)

```

1.3.2 Distributions

Distribution des classes de plantes :

* Nous calculons le nombre d'images par échantillon et par plante.

```
[62]: plant_count = df_disease.groupby(['dataset', 'sample', 'plant_fr'],  
    ↪as_index=False).file.count()  
# Les 33 images de l'échantillon de test du dataset new-plant-diseases-dataset  
↪n'ont pas de classe, elles sont ignorées  
plant_count = pd.  
    ↪pivot_table(plant_count[~((plant_count['dataset']=='new-plant-diseases-dataset')  
    ↪& (plant_count['sample']=='test'))], index='plant_fr', columns=['dataset',  
    ↪'sample'], aggfunc='sum').reset_index()
```

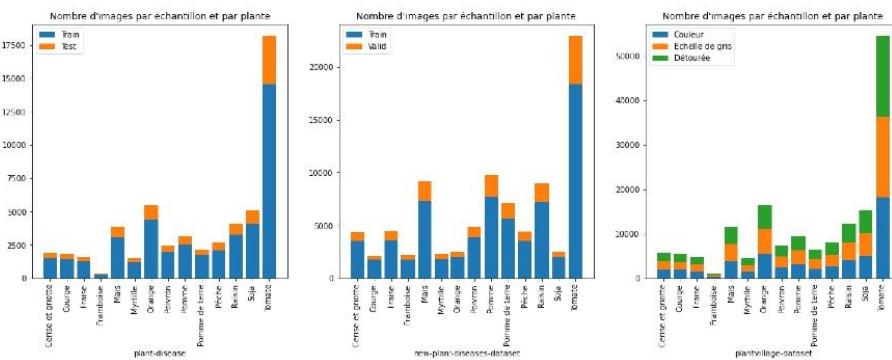
- Nous affichons la distribution dans 3 graphiques (un par jeu de données)

```
[63]: import matplotlib.pyplot as plt  
  
fig = plt.figure(figsize=(20, 6))  
ax1 = fig.add_subplot(131)  
ax1.bar(range(len(plant_count)), plant_count[['file',  
    ↪'plant-disease','train']], label='Train')  
ax1.bar(range(len(plant_count)), plant_count[['file', 'plant-disease','test']],  
    ↪bottom = plant_count[['file', 'plant-disease','train']], label='Test')  
ax1.set_xticks(plant_count.index)  
ax1.set_xticklabels(plant_count.plant_fr.values, rotation=90)  
ax1.set_xlabel('plant-disease')  
ax1.set_title("Nombre d'images par échantillon et par plante")  
ax1.legend()  
  
ax2 = fig.add_subplot(132)  
ax2.bar(range(len(plant_count)), plant_count[['file',  
    ↪'new-plant-diseases-dataset','train']], label='Train')  
ax2.bar(range(len(plant_count)), plant_count[['file',  
    ↪'new-plant-diseases-dataset','valid']], bottom = plant_count[['file',  
    ↪'new-plant-diseases-dataset','train']], label='Valid')  
ax2.set_xticks(plant_count.index)  
ax2.set_xticklabels(plant_count.plant_fr.values, rotation=90)  
ax2.set_xlabel('new-plant-diseases-dataset')  
ax2.set_title("Nombre d'images par échantillon et par plante")  
ax2.legend()  
  
ax3 = fig.add_subplot(133)  
ax3.bar(range(len(plant_count)), plant_count[['file',  
    ↪'plantvillage-dataset','color']], label='Couleur')  
ax3.bar(range(len(plant_count)), plant_count[['file',  
    ↪'plantvillage-dataset','grayscale']], bottom = plant_count[['file',  
    ↪'plantvillage-dataset','color']], label='Echelle de gris')
```

```

ax3.bar(range(len(plant_count)), plant_count[['file', u
    ↵'plantvillage-dataset', 'segmented']], bottom = plant_count[['file', u
    ↵'plantvillage-dataset', 'color']] + plant_count[['file', u
    ↵'plantvillage-dataset', 'grayscale']], label='Détourée')
ax3.set_xticks(plant_count.index)
ax3.set_xticklabels(plant_count.plant_fr.values, rotation=90)
ax3.set_xlabel('plantvillage-dataset')
ax3.set_title("Nombre d'images par échantillon et par plante")
ax3.legend()
plt.show();

```



Distribution des classes de maladies :

* Nous calculons le nombre d'images par échantillon et par maladie.

```

[64]: disease_count = df_disease.groupby(['dataset', 'sample', 'disease_fr'], u
    ↵as_index=False).file.count()
# Les 33 images de l'échantillon de test du dataset new-plant-diseases-dataset u
# n'ont pas de classe, elles sont ignorées
disease_count = pd.
pivot_table(disease_count[~((disease_count['dataset']=='new-plant-diseases-dataset') u
    & (disease_count['sample']=='test'))], index='disease_fr', u
    & columns=['dataset', 'sample'], aggfunc='sum').reset_index()

```

- Nous affichons la distribution dans 3 graphiques (un par jeu de données)

```

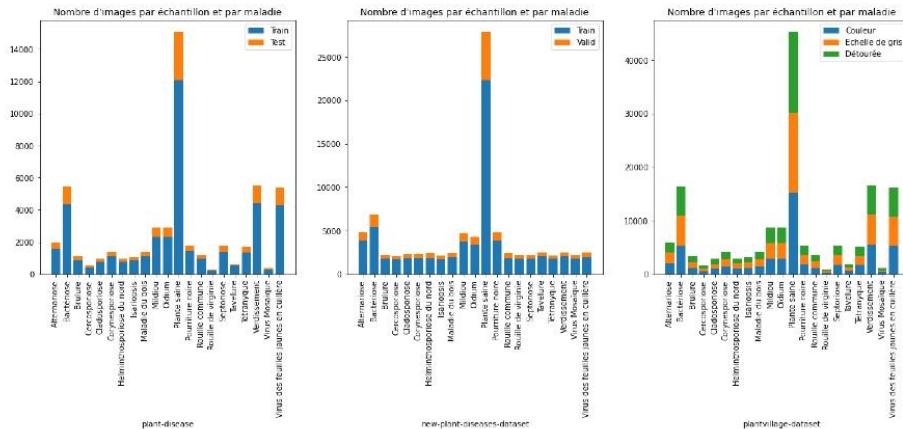
[65]: fig = plt.figure(figsize=(20, 6))
ax1 = fig.add_subplot(131)
ax1.bar(range(len(disease_count)), disease_count[['file', u
    ↵'plant-disease', 'train']], label='Train')
ax1.bar(range(len(disease_count)), disease_count[['file', u
    ↵'plant-disease', 'test']], bottom = disease_count[['file', u
    ↵'plant-disease', 'train']], label='Test')

```

```
ax1.set_xticks(disease_count.index)
ax1.set_xticklabels(disease_count.disease_fr.values, rotation=90)
ax1.set_xlabel('plant-disease')
ax1.set_title("Nombre d'images par échantillon et par maladie")
ax1.legend()

ax2 = fig.add_subplot(132)
ax2.bar(range(len(disease_count)), disease_count[['file', u
    ↪ 'new-plant-diseases-dataset', 'train']], label='Train')
ax2.bar(range(len(disease_count)), disease_count[['file', u
    ↪ 'new-plant-diseases-dataset', 'valid']], bottom = disease_count[['file', u
    ↪ 'new-plant-diseases-dataset', 'train']], label='Valid')
ax2.set_xticks(disease_count.index)
ax2.set_xticklabels(disease_count.disease_fr.values, rotation=90)
ax2.set_xlabel('new-plant-diseases-dataset')
ax2.set_title("Nombre d'images par échantillon et par maladie")
ax2.legend()

ax3 = fig.add_subplot(133)
ax3.bar(range(len(disease_count)), disease_count[['file', u
    ↪ 'plantvillage-dataset', 'color']], label='Couleur')
ax3.bar(range(len(disease_count)), disease_count[['file', u
    ↪ 'plantvillage-dataset', 'grayscale']], bottom = disease_count[['file', u
    ↪ 'plantvillage-dataset', 'color']], label='Echelle de gris')
ax3.bar(range(len(disease_count)), disease_count[['file', u
    ↪ 'plantvillage-dataset', 'segmented']], bottom = disease_count[['file', u
    ↪ 'plantvillage-dataset', 'color']] + disease_count[['file', u
    ↪ 'plantvillage-dataset', 'grayscale']], label='Détourée')
ax3.set_xticks(disease_count.index)
ax3.set_xticklabels(disease_count.disease_fr.values, rotation=90)
ax3.set_xlabel('plantvillage-dataset')
ax3.set_title("Nombre d'images par échantillon et par maladie")
ax3.legend()
plt.show();
```



1.3.3 Distribution des classes par plantes et par maladies :

- Nous calculons le nombre d'images par échantillon, par plante et par maladie pour le jeu de données new-plant-diseases-dataset

```
[66]: plant_disease_count = pd.read_csv('new-plant-diseases-dataset.csv')
       .groupby(['sample', 'plant_fr', 'disease_fr'], as_index=False).file.count()
# Les 33 images de l'échantillon de test du dataset new-plant-diseases-dataset
# n'ont pas de classe, elles sont ignorées
plant_disease_count = pd.read_csv('new-plant-diseases-dataset.csv')
       .pivot_table(plant_disease_count[~(plant_disease_count['sample']=='test')], 
       index=['plant_fr', 'disease_fr'], columns=['sample'], aggfunc='sum')
       .reset_index()
plant_disease_count['total'] = plant_disease_count[['file', 'train']] +
       plant_disease_count[['file', 'valid']]
plant_disease_count['order'] = plant_disease_count.disease_fr.apply(lambda x: 0 if x == 'Plante saine' else 1)
plant_disease_count.sort_values(by=['plant_fr', 'order', 'disease_fr'],
       inplace=True)
plant_disease_count.reset_index(inplace=True)
plant_disease_count.drop('index', axis=1, inplace=True)
```

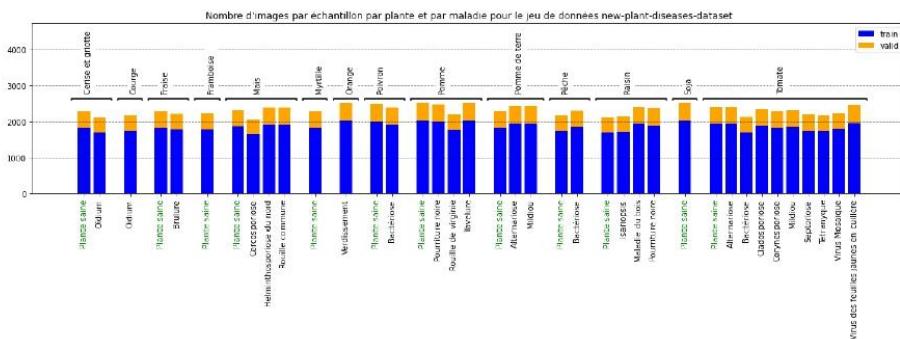
- Nous affichons la distribution dans un graphique :
 - Les maladies (en abscisse) sont regroupées par plante (en commentaire).
 - La distribution de la plante saine est toujours affichée en premier (lorsqu'elle existe)
 - Les groupes de maladie sont espacés pour une meilleure lisibilité

```
[67]: fig = plt.figure(figsize=(20, 4))
# On augmente artificiellement la dimension des ordonnées pour laisser de la place pour les noms des plantes
ymax = plant_disease_count.total.max() + 2200
plt.ylim([0, ymax])
# On initialise les variables utilisées par la suite
previous = plant_disease_count.iloc[0, 0] # On mémorise la plante précédente dans l'ordre d'apparition dans le dataset
pos = 0 # On mémorise la position de l'étiquette d'abscisse à traiter
start = -1 # On mémorise l'abscisse de démarrage de la première accolade
yline = plant_disease_count.total.max() + 100 # On mémorise l'ordonnée des accolades
xoffset = 0.2 # On gère un décroché sur l'axe des abscisses pour l'accordéon
yoffset = 50 # On gère un décroché sur l'axe des ordonnées pour l'accordéon
Xtick = [] # On initialise la liste permettant de gérer la position des étiquettes en abscisse
Xlabel = [] # On initialise la liste permettant de gérer le libellé des étiquettes en abscisse
XSpecial = [] # On initialise la liste permettant de gérer la position des étiquettes spéciales en abscisse (plante saine)
label1 = '' # On initialise la légende de la barre des données d'entraînement
label2 = '' # On initialise la légende de la barre des données de validation
# Principe : on parcourt le dataset, lorsqu'on change de plante :
# - On crée un espace entre groupe
# - On affiche une accolade au dessus des barres avec le nom de la plante
for i in range(len(plant_disease_count)):
    plant, disease, train, valid, total, order = plant_disease_count.iloc[i, :]
    if plant != previous: # Détection d'un changement de plante
        # Création de l'accordéon
        plt.plot([start + xoffset, start + xoffset, pos - xoffset, pos - xoffset], [yline - yoffset, yline, yline, yline - yoffset], color='black')
        # Affichage du nom de la plante
        plt.text(start + (pos-start)//2, yline + 200, previous, rotation=90, backgroundcolor='white')
        previous = plant
        start = pos
        pos+=1
    # Gestion des étiquettes en abscisse
    Xtick.append(pos) # On mémorise la position de l'étiquette
    Xlabel.append(disease) # On mémorise le libellé de l'étiquette
    if order == 0:
        XSpecial.append(i) # On mémorise les étiquettes 'Plante saine'
    # Pour la dernière maladie, on affiche une légende
    if i == len(plant_disease_count)-1:
        label1 = 'train'
        label2 = 'valid'
```

```

plt.bar([pos],[train], color=['blue'], label=label1)
plt.bar([pos],[valid], bottom=[train], color=['orange'], label=label2)
pos +=1
# Gestion de la dernière accolade
plt.plot([start + xoffset, start + xoffset, pos - xoffset, pos - xoffset], [yline - yoffset, yline, yline, yline - yoffset], color='black')
plt.text(start + (pos-start)//2, yline + 200, previous, rotation=90, backgroundcolor='white')
# Gestion de l'affichage des étiquettes en abscisse
plt.xticks(Xtick, Xlabel, rotation=90)
for i in XSpecial:
    plt.gca().get_xticklabels()[i].set_color("green")
plt.legend()
plt.grid(True, linestyle = '--', axis='y', color='black', alpha=0.5)
plt.title("Nombre d'images par échantillon par plante et par maladie pour le jeu de données new-plant-diseases-dataset");

```



1.3.4 Sélection

Le jeu de données **new-plant-diseases-dataset** est prometteur aussi bien en volume qu'en organisation.

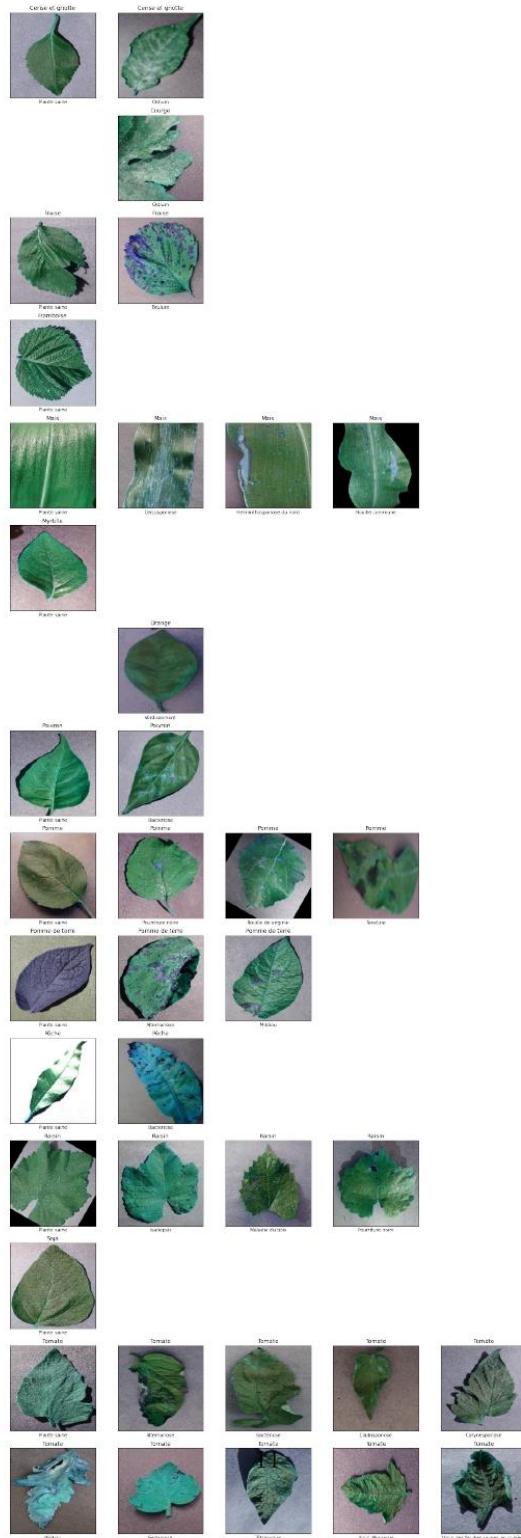
Affichons quelques images aléatoirement dans l'échantillon d'entraînement.

```
[68]: # Préparation du jeu de données dans lequel nous allons tirer une photo
      ↴ aléatoirement pour chaque classe
img_random = df_disease[(df_disease['dataset']=='new-plant-diseases-dataset') &
      ↴ (df_disease['sample']=='train')].copy()
img_random['malade'] = img_random.disease_fr.apply(lambda x: 0 if x=='Plante
      ↴ saine' else 1)
img_random.sort_values(by=['plant_fr', 'malade', 'disease_fr'], ascending=True,
      ↴ inplace=True)
img_random.reset_index(inplace = True)
```

```
    img_random.drop('index', axis=1, inplace=True)
```

```
[69]: # Préparation de la grille d'affichage des images sélectionnées aléatoirement
# Les plantes sont présentées en ligne par ordre alphabétique
# La première colonne présente toujours une plante saine, les autres colonnes
# présentent les maladies
# La grille contient 5 colonnes et 15 lignes (1 ligne par plante et 2 lignes
# pour la tomate qui a plus de 5 classes de maladie)
import numpy as np
plant_list = img_random.plant_fr.unique()
img_grid = np.zeros((len(plant_list)+1, 5), np.int)
for row, plant in enumerate(plant_list):
    disease_list = img_random[img_random['plant_fr']==plant].disease_fr.unique()
    offset = not('Plante saine' in disease_list)
    for col, disease in enumerate(disease_list):
        colnum = col + offset
        rownum = row
        if colnum >= 5: # Pour les tomates, il y a plus de 5 classes de maladie
            colnum -= 5
            rownum = row + 1
        img_grid[rownum, colnum] = np.random.
        choice(img_random[(img_random['plant_fr']==plant) &
        (img_random['disease_fr']==disease)].index)
```

```
[73]: # Affichage de la grille d'images aléatoires
import cv2
fig = plt.figure(figsize=(20, 60))
maxrow = img_grid.shape[0]
maxcol = img_grid.shape[1]
for posx, row in enumerate(img_grid):
    for posy, item in enumerate(row):
        if item !=0:
            pos = posx * maxcol + posy + 1
            ax = fig.add_subplot(maxrow, maxcol, pos)
            ax.set_xticks([])
            ax.set_yticks([])
            ax.set_xlabel(img_random.iloc[item, 13])
            ax.set_title(img_random.iloc[item, 12])
            fname = racine + ossep + img_random.iloc[item, 0] + ossep +
            img_random.iloc[item, 1] + ossep + img_random.iloc[item, 2]
            if os.name == 'nt': # Pour tenir compte de la limite de 260
            # caractères dans le nom d'un fichier imposée par Windows
                fname = "\\\\" + fname
            img = cv2.imread(fname, cv2.IMREAD_UNCHANGED)
            ax.imshow(img)
```



1.3.5 Images moyennes

Prenons toutes les images des échantillons d'entraînement et de validation des différentes classes (plante et maladie) et réalisons des images moyennes.

Compte tenu du nombre d'images à traiter, nous utiliserons le multi-threading pour calculer et afficher les images moyennes.

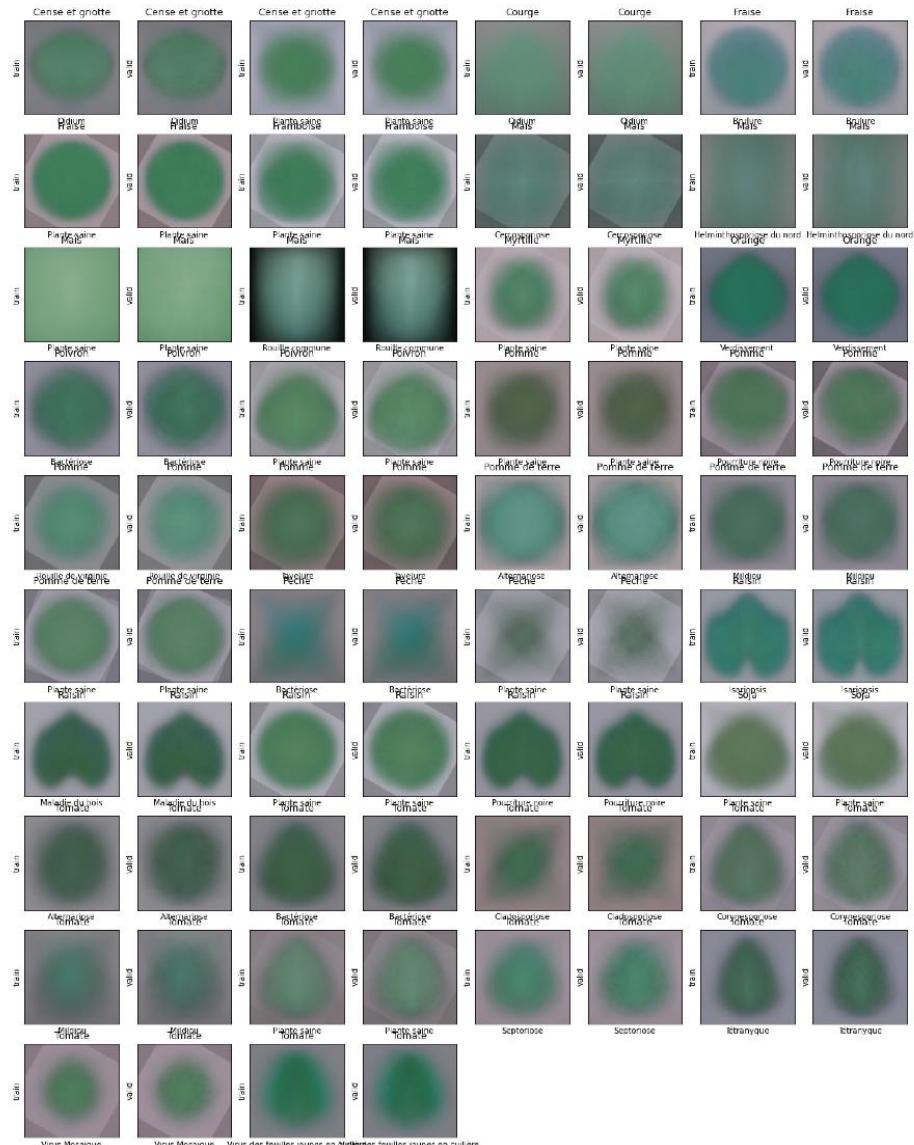
```
[74]: import threading
Lock = threading.RLock()

# Création de la classe dérivée de Thread traitant des lots d'images (par type ↴
# d'échantillon, par plante et par maladie)
class DisplayImageMean(threading.Thread):
    def __init__(self, df, plant, disease, sample, fig, pos):
        threading.Thread.__init__(self)
        self.df = df
        self.plant = plant
        self.disease = disease
        self.sample = sample
        self.fig = fig
        self.pos = pos
    # Calcul de l'image moyenne pour un lot (pour un échantillon, une plante et ↴
    # une maladie)
    def get_image_mean(self):
        lot = self.df[(self.df['sample']==self.sample) & (self.
        ↪df['plant_fr']==self.plant) & (self.df['disease_fr']==self.disease)]
        img_mean = np.zeros((256, 256, 3), np.int)
        fileNames = (lot.iloc[:, 0] + ossep + lot.iloc[:, 1] + ossep + lot.
        ↪iloc[:, 2]).values
        for f in fileNames:
            f = racine + ossep + f
            if os.name == 'nt': # Pour tenir compte de la limite de 260 ↴
            # caractères dans le nom d'un fichier imposée par Windows
            f = "\\\?\\" + f
            img = cv2.imread(f)
            img_mean += img
        img_mean /= len(fileNames)
        return img_mean
    # Affichage de l'image moyenne d'un lot
    def format_image_mean(self, img):
        ax = self.fig.add_subplot(10, 8, self.pos)
        ax.set_xticks([])
        ax.set_yticks([])
        ax.set_xlabel(self.disease)
```

```
    ax.set_ylabel(self.sample)
    ax.set_title(self.plant)
    ax.imshow(img)

def run(self):
    image = self.get_image_mean()
    with Lock:
        self.format_image_mean(image)

[75]: %%time
import matplotlib.pyplot as plt
import numpy as np
import cv2
# Préparation des lots d'image
case_list = df_disease[(df_disease['dataset']=='new-plant-diseases-dataset') &
    ~(df_disease['sample']!='test')].groupby(['plant_fr', 'disease_fr', u
    'sample']).file.count().reset_index()
fig = plt.figure(figsize=(20, 26))
threads = []
# Lancement des threads sur chaque lot
for i in range(len(case_list)):
    plant, disease, sample = case_list.iloc[i, :-1]
    DIM = DisplayImageMean(df_disease, plant, disease, sample, fig, i+1)
    threads.append(DIM)
    DIM.start()
for t in threads:
    t.join()
plt.show(); # 5 minutes et 16 secondes (contre plus de 24 minutes en traitement u
    u séquentiel)
```



Wall time: 7min 11s

Les images moyennes présentent des similitudes : * un halo verdâtre centré (sauf pour les plantes saines de maïs) * les contours du halo sont plus sombres * le fond est gris * certaines images semblent avoir subi une rotation d'une vingtaine de degrés

10.2. Réduction de dimension sur les images

reduction_dimensions

December 25, 2020

1 Réduction de la dimension des données par PCA (Principal Components Analysis)

1.1 Importation des modules nécessaires pour le travail

```
[1]: import cv2
import numpy as np
from tqdm import tqdm
from sklearn.decomposition import PCA
import matplotlib.pyplot as plt
from matplotlib import offsetbox
```

1.2 Récupération et stockage des fichiers

Pour le travail à réaliser, un dictionnaire sera créer afin de stocker le nom des classes, leurs images correspondantes, les images réduites par PCA et les composants PCA utilisés pour cette réduction.

```
[4]: ### On importe les outils nécessaires à la récupéraion des images et du nom des
      ↵classes
      import os
      import sys
      sys.path.append(os.path.abspath(os.path.join(os.getcwd(), os.pardir)))

      from lib import ressources as res
      from lib import tools

      racine = os.path.abspath(os.path.realpath(res.dir_root))
```

```
[5]: ### Le nom des classes sont récupérés et stocké dans le dictionnaire sous la
      ↵clé 'Classe'
      classes=tools.listclasses(res.dir_dataset_train)
      data={'Classe':[],'Listes_images':[]}
      data['Classe']=classes

      ### Pour chaque classe, une liste d'images sous forme de tableau à 1 dimension
      ↵est ajouté sous la clé 'Listes_images'
      for i in range(len(classes)):
          liste_images=[]
```

```

for path in tools.listdir(res.dir_dataset_train+os.sep+classes[i]):
    img=cv2.imread(path)
    img=cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    img=img.reshape(img.shape[0]*img.shape[1]*img.shape[2])
    liste_images.append(img)
data['Listes_images'].append(np.array(liste_images))

```

1.3 Réduction de la dimension des images par PCA

Lors de cette étape, nous allons opérer une réduction de dimension par PCA en indiquant une conservation de variance de 95%.

- Réalisation de la réduction de dimension

Initialisation de constantes utilisées dans la suite du code

```

[6]: ### Dimensions des photos
height=256
length=256
n_pixel=3

### Nombre de classe, implémenter 38 pour avoir la totalité des classes du
# dataset
n_classe=6

```

```

[7]: ### Initiation de la méthode PCA, on conserve 95% de la variance
pca=PCA(0.95)

### Les images réduites vont être stockées dans le dictionnaire sous la clé
# 'Listes_images_pca'
### Les composants ayant servi à la réduction vont être sauvegardés sous la clé
# 'Components'
data['Listes_images_pca']=[]
data['Components']=[]

### La réduction de dimension est faite par classe
for liste_images in tqdm(data['Listes_images'][:n_classe]):
    liste_images_pca=pca.fit_transform(liste_images)
    data['Listes_images_pca'].append(liste_images_pca)
    data['Components'].append(pca.components_)

```

100% | 6/6 [28:58<00:00, 289.69s/it]

- Quantification de la PCA

```

[8]: ### Création d'une liste avec les pourcentages de diminution de dimensions
# obtenue par PCA pour chaque classe
n_components=np.array([])

```

```

for components in data['Components']:
    n_components=np.append(n_components,1-(len(components)/
    ↴(height*length*n_pixel)))

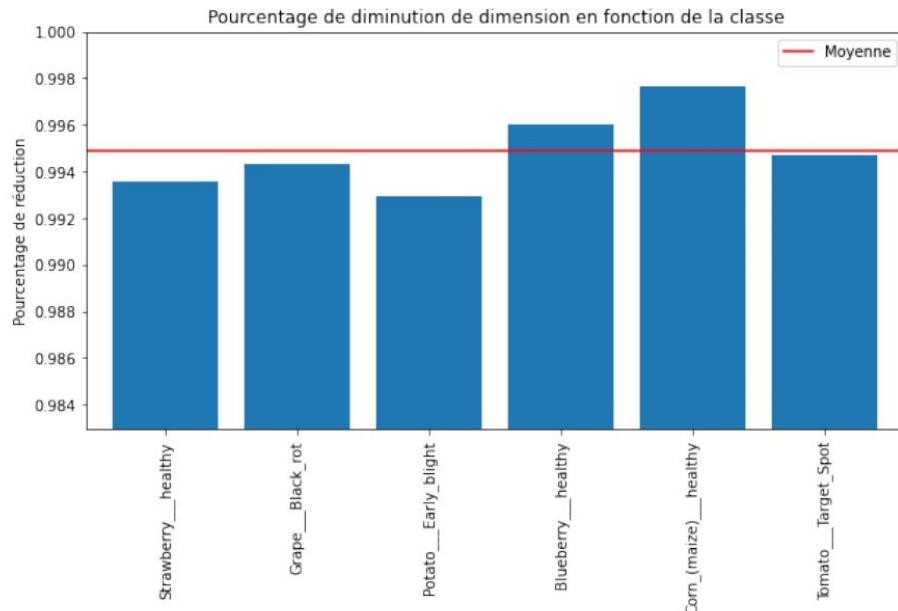
range_classes=range(1,n_classe+1)
moyenne=n_components.mean()

### On réalise un barplot pour observer la réduction de dimension selon les ↴
classes
### Un trait rouge vertical est ajouté pour déterminer la réduction moyenne
print("Remarque: l'échelle des ordonnées est coupée pour une meilleure ↴
observation des différences entre classe.")
plt.figure(figsize=(10,5))
plt.plot([0,n_classe+1],[moyenne,moyenne],color='red',label='Moyenne')
plt.bar(range_classes,n_components)
plt.xticks(range_classes,data['Classe'][:n_classe],rotation=90)
plt.ylabel('Pourcentage de réduction')
plt.title('Pourcentage de diminution de dimension en fonction de la classe')
plt.ylim(n_components.min()-0.01,1)
plt.xlim(0.4,n_classe+0.6)
plt.legend()
plt.show()

print('Pourcentage moyen de réduction de dimension :',np.
    ↴round(moyenne*100,3),'%','\n')
print("La méthode PCA a permis de diminuer grandement la dimension des données, ↴
avec un réduction de plus de 99% pour une perte de seulement 5% de la ↴
variance.")

```

Remarque: l'échelle des ordonnées est coupée pour une meilleure observation des différences entre classe.



Pourcentage moyen de réduction de dimension : 99.486 %

La méthode PCA a permis de diminuer grandement la dimension des données, avec un réduction de plus de 99% pour une perte de seulement 5% de la variance.

1.4 Affichage des données sur un graphique bi-dimensionnel

Pour afficher ces graphiques, nous allons prendre en compte les deux premiers composants des images réduites par PCA. Ils permettent souvent d'expliquer la majeure partie de la variance des données. Nous allons visualiser la répartition des images par classe, en affichant certaines images pour avoir une meilleure compréhension de cette répartition.

```
[9]: ### Définition d'une fonction permettant de visualiser les données sur un
      ↵ graphique de projection bi-dimensionnel.
      ### Cette fonction permet d'afficher des images correspondant à certains points
      ↵ du graphique.
      ### Le paramètre thumb_frac permet de jouer sur la quantité d'images à afficher
      ↵ sur le graphique.
      ### Le paramètre zoom permet de changer la taille des images affichées.

def plot_components(data, images=None, ax=None,
                    thumb_frac=0.09, zoom=0.2):
    ax = ax or plt.gca()
```

```

ax.plot(data[:, 0], data[:, 1], '.b')
if images is not None:
    min_dist_2 = (thumb_frac * max(data.max(0)[:, 2] - data.min(0)[:, 2])) ** 2
    shown_images = np.array([2 * data.max(0)[:, 2]])
    for i in range(data.shape[0]):
        dist = np.sum((data[i, :2] - shown_images) ** 2, 1)
        if np.min(dist) < min_dist_2:
            # On ne montre pas les points trop proches
            continue
        shown_images = np.vstack([shown_images, data[i, :2]])
    imagebox = offsetbox.AnnotationBbox(
        offsetbox.OffsetImage(images[i], zoom=zoom),
        data[i, :2])
    ax.add_artist(imagebox)

```

```

[10]: print("L'observation de la dispersion des images nous donne certains\u202a
       informations sur le jeu de données.",\n
       "Certaines images ont été tournées pour augmenter les données.",\n
       "On observe que dans beaucoup de cas un classement se fait selon la\u202a
       'luminosité' des photos.",\n
       "On remarque ainsi que certaines photos sont très surexposées et des\u202a
       photos sont segmentées avec la feuille seule sur un fond noir.")

```

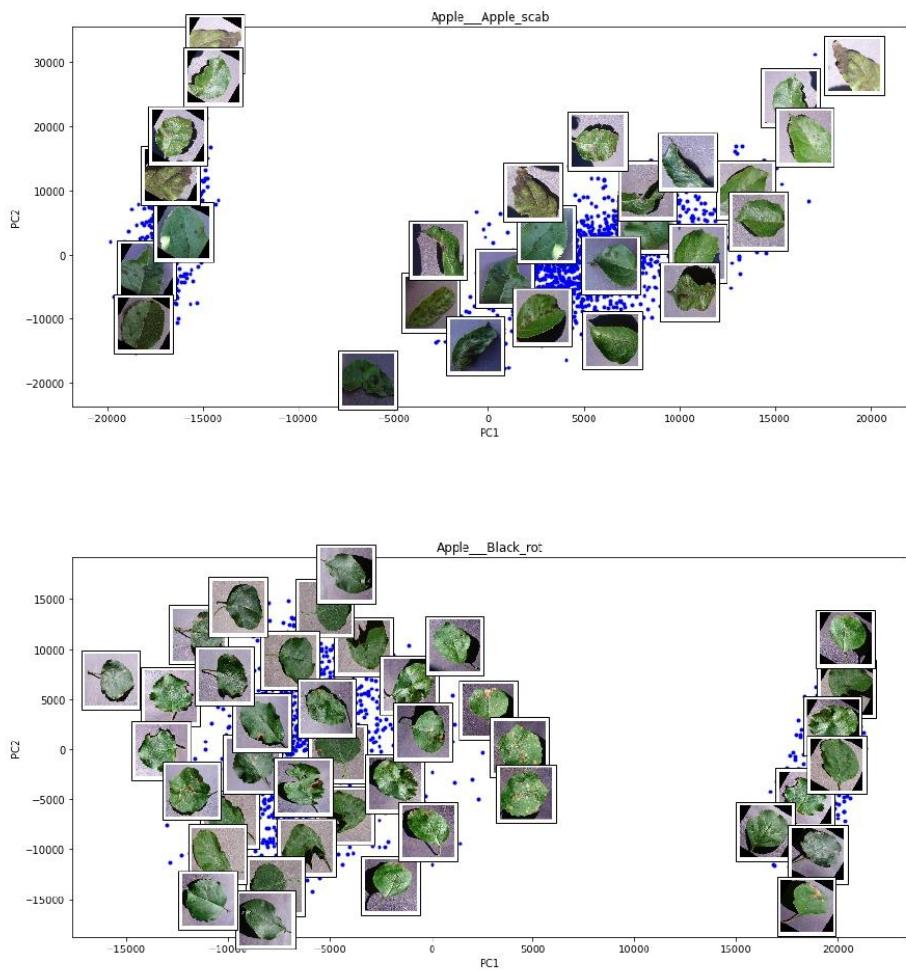
```

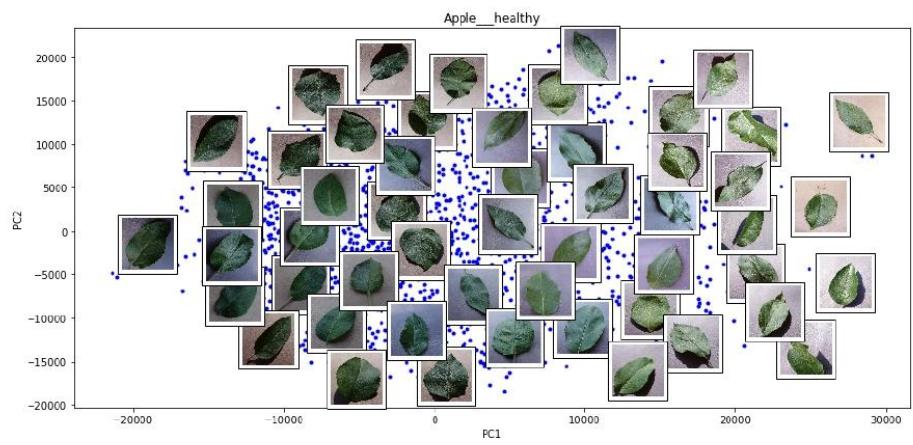
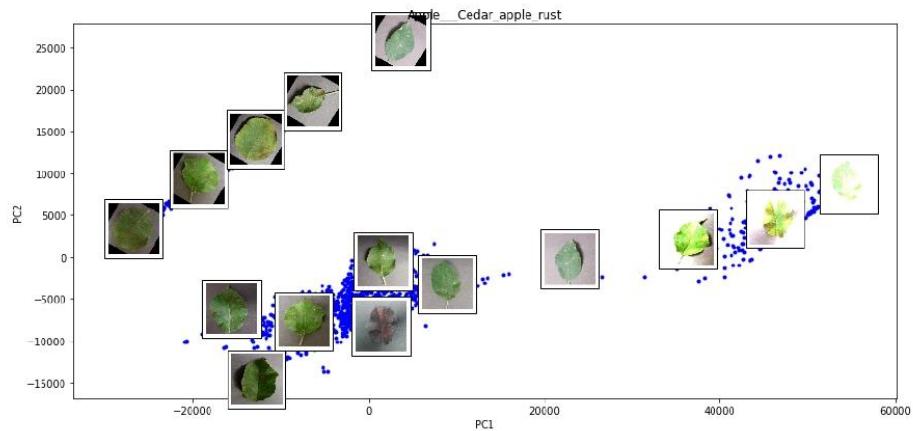
### Affichage des graphiques par classe

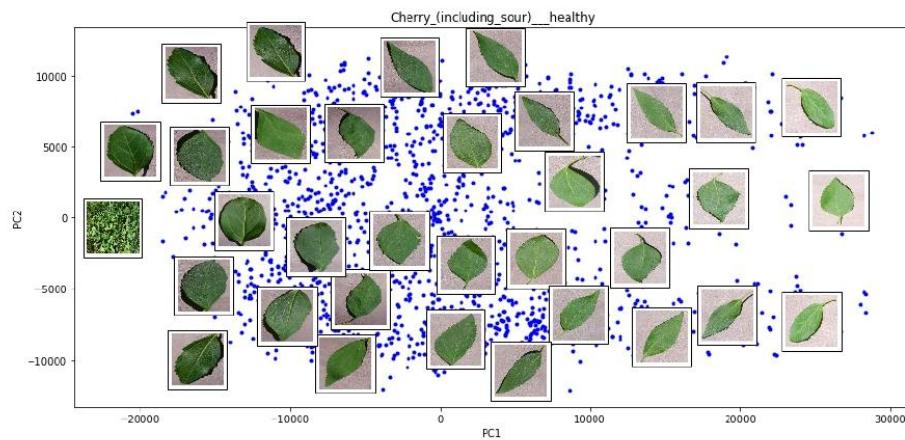
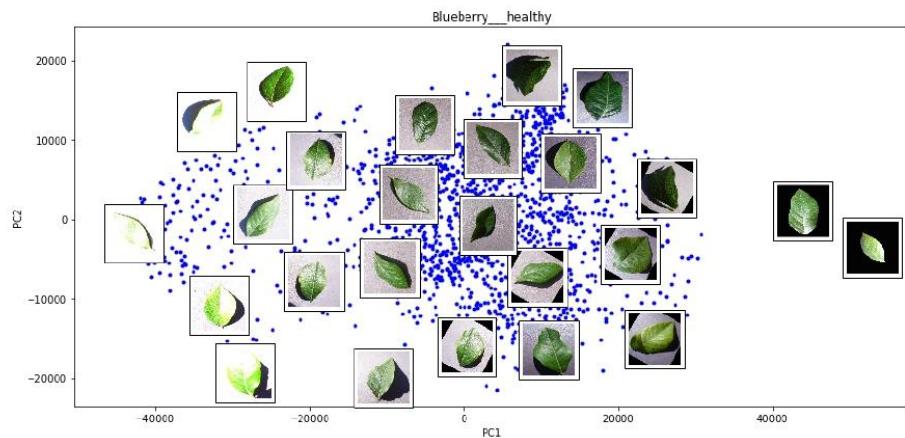
for liste_images_pca, liste_images, name in zip\u202a
    (data['Listes_images_pca'], data['Listes_images'][:n_classe], data['Classe']):
    ### Redimensionnement des images pour leur affichage
    images=[]
    for image in liste_images:
        image=image.reshape(height,length,n_pixel)
        images.append(image)
    ### Définition des graphiques
    plt.figure(figsize=(15,7))
    plt.xlabel('PC1')
    plt.ylabel('PC2')
    plt.title(name)
    plot_components(liste_images_pca,images)

```

L'observation de la dispersion des images nous donne certains informations sur le jeu de données. Certaines images ont été tournées pour augmenter les données. On observe que dans beaucoup de cas un classement se fait selon la 'luminosité' des photos. On remarque ainsi que certaines photos sont très surexposées et des photos sont segmentées avec la feuille seule sur un fond noir.







10.3. CNN VGG16 & Tranfer learning

cnn_vgg_16_augmented

December 25, 2020

```
[1]: # les images vont être ajustées par le CNN de 256*256 à image_size*image_size
image_size = 256

# nombre epochs à réaliser par itération
n_epochs = 0

# nombre de classes
n_class = 39

# plateforme colab
is_colab_platform = False

[2]: # ! rm -rf "/content/dataset"

[3]: if (is_colab_platform == True):

    from google.colab import drive
    drive.mount('/content/drive', force_remount=True)

[4]: import os

# nom du cnn
cnn_name = 'cnn_vgg16_augmented'

if (is_colab_platform == True):

    dataset_train = "dataset/New Plant Diseases Dataset(Augmented)/New Plant\u20acDiseases Dataset(Augmented)/train/"
    dataset_valid = "dataset/New Plant Diseases Dataset(Augmented)/New Plant\u20acDiseases Dataset(Augmented)/valid/"
    dataset_test = "dataset/new Plant Diseases Dataset(Augmented)/New Plant\u20acDiseases Dataset(Augmented)/test/"

    racine = os.sep + 'content'

    weights_path = racine + os.sep + 'drive' + os.sep + 'MyDrive' + os.sep + \
        cnn_name + '_' + str(image_size) + '_weights'
```

```

history_path = racine + os.sep + 'drive' + os.sep + 'MyDrive' + os.sep +_
↪cnn_name + '_' + str(image_size) + '_history'

else:

    sys.path.append(os.path.abspath(os.path.join(os.getcwd(), os.pardir)))

    from lib import ressources as res
    racine = os.path.abspath(os.path.realpath(res.dir_root))

    weights_path = res.dir_data + os.sep + cnn_name + '_' + str(image_size) +_
↪'_weights'
    print(weights_path)

    history_path = res.dir_data + os.sep + cnn_name + '_' + str(image_size) +_
↪'_history'
    print(history_path)

    dataset_train = res.dir_dataset_train
    dataset_valid = res.dir_dataset_valid
    dataset_test = res.dir_dataset_test

```

```

C:\Users\NOEL\dev\python\DATA
SCIENTIST\_projet\data\cnn_vgg16_augmented_256_weights
C:\Users\NOEL\dev\python\DATA
SCIENTIST\_projet\data\cnn_vgg16_augmented_256_history

```

```

[5]: import sys

# Follow instructions for using kaggle with colab : https://www.kaggle.com/
↪general/74235
if (is_colab_platform == True):

    # Extract the json api credential from kaggle
    # Load it in next cell
    ! pip install -q kaggle
    from google.colab import files
    files.upload()

    # Create a kaggle folder
    # Copy the json credential to it.
    ! mkdir ~/.kaggle
    ! cp kaggle.json ~/.kaggle/

    # Change the permissions of the file.
    ! chmod 600 ~/.kaggle/kaggle.json

```

```
# Download the dataset zip
! kaggle datasets download -d vippooooool/new-plant-diseases-dataset

# unzip the dataset zip
! mkdir dataset
! unzip new-plant-diseases-dataset.zip -d dataset

# !dir

# os.listdir(dataset_train)

[6]: if (is_colab_platform == True):

    dataset_other = racine + os.sep + 'drive' + os.sep + 'MyDrive' + os.sep +_
    ↴ 'dataset_other.zip'

    # ! rm -Rf "/content/dataset"
    ! unzip '/content/drive/MyDrive/dataset_other.zip' -d "/content/dataset/New_"
    ↴ Plant Diseases Dataset(Augmented)/New Plant Diseases Dataset(Augmented)/"
```

1 Méthodes relatives au CNN

1.1 Création du CNN

1.2 Sauvegarde des données

1.3 Augmentation du jeu de données

```
[7]: from keras.applications.vgg16 import preprocess_input
from keras.preprocessing.image import ImageDataGenerator

batch_size = 32

from keras.applications.vgg16 import preprocess_input

train_data_generator = ImageDataGenerator(
    preprocessing_function = preprocess_input,
    # data augmentation
    rotation_range = 10,
    width_shift_range = 0.1,
    height_shift_range = 0.1,
    zoom_range = 1.1,
    horizontal_flip = True
```

```

    )

test_data_generator = ImageDataGenerator(
    preprocessing_function = preprocess_input)

train_set = train_data_generator.flow_from_directory(dataset_train,
    target_size = (image_size , image_size),
    batch_size = batch_size)

test_set = test_data_generator.flow_from_directory(dataset_valid,
    target_size = (image_size , image_size),
    batch_size = batch_size)

```

Found 72337 images belonging to 39 classes.
 Found 18082 images belonging to 39 classes.

```
[8]: # l'historique d'entraînement est concaténé avec les précédents résultats
def concatenate_histories(previous_history, training_history):

    train_accuracy = previous_history['accuracy'] + training_history['accuracy']
    val_accuracy = previous_history['val_accuracy'] +_
    ↪training_history['val_accuracy']
    train_loss = previous_history['loss'] + training_history['loss']
    val_loss = previous_history['val_loss'] + training_history['val_loss']

    history = {
        'accuracy': train_accuracy,
        'val_accuracy': val_accuracy,
        'loss': train_loss,
        'val_loss': val_loss
    }

    # retourne l'historique mis à jour
    return history
```

```
[9]: import pickle
from shutil import copyfile
import fnmatch

# enregistre l'historique de l'entraînement et les poids du CNN
def save_data(model, new_history=None):

    # l'history est mis à jour à partir de training_history et, le cas échéant, ↪
    ↪de l'historique
    # de l'entraînement précédent qui a été enregistré
```

```

is_trained = os.path.isfile(history_path)

# le CNN a déjà été entrainé : un historique d'entraînement existe déjà
if (is_trained == True):

    previous_history = pickle.load(open(history_path, 'rb'))

    # cette fois, le CNN n'a pas été entraîné
    if (new_history == None):
        history = previous_history
    # cette fois-ci, le cnn a été entraîné
    else:
        history = concatenate_histories(previous_history, new_history)

    # c'est le premier entraînement du CNN
else:
    history = new_history

    # history est enregistré
    pickle.dump(history, open(history_path, 'wb'))

    # les poids sont enregistrés
model.save_weights(weights_path);

return history

```

```

[10]: # le cnn est entraîné
def training(model, n_epochs):

    steps_per_epoch = train_set.n // train_set.batch_size
    validation_step = test_set.n // test_set.batch_size

    training_history = model.fit_generator(generator=train_set,
                                            epochs = n_epochs,
                                            steps_per_epoch = steps_per_epoch,
                                            validation_data = test_set,
                                            validation_steps = validation_step)

    if (n_epochs == 0):
        history = None
    else:
        history = training_history.history

    # les données sont sauvegardées
    # en présence d'entraînement précédents, les historiques sont concaténés
    ↵ avec sauvegarde
    history_total = save_data(model, history)

```

```
    return history_total
```

2 Entrainement du CNN

```
[11]: # Import des librairies pour le CNN
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout, GlobalAveragePooling2D
from keras.applications.vgg16 import VGG16
from keras.optimizers import Adam
```

```
[12]: # Modèle VGG16
base_model = VGG16(weights='imagenet', include_top=False)

# freeze les couches du VGG16
for layer in base_model.layers:
    layer.trainable = False

model = Sequential()
model.add(base_model) # Ajout du modèle VGG16
model.add(GlobalAveragePooling2D())
model.add(Dense(1024, activation='relu'))
model.add(Dropout(rate=0.2))
model.add(Dense(512, activation='relu'))
model.add(Dropout(rate=0.2))
model.add(Dense(n_class, activation='softmax'))

# le cnn est compilé
model.compile(optimizer='adam', loss='categorical_crossentropy',
              metrics=['accuracy'])

# la structure du modèle est affichée
model.summary()

# le CNN est entraîné
for i in range(0, n_epochs):
    training(model, 1)
```

Model: "sequential"

| Layer (type) | Output Shape | Param # |
|-------------------------------------------|-------------------------|----------|
| <hr/> | | |
| vgg16 (Functional) | (None, None, None, 512) | 14714688 |
| <hr/> | | |
| global_average_pooling2d (G1 (None, 512)) | | 0 |
| <hr/> | | |

```

dense (Dense)           (None, 1024)      525312
-----
dropout (Dropout)       (None, 1024)      0
-----
dense_1 (Dense)         (None, 512)       524800
-----
dropout_1 (Dropout)     (None, 512)       0
-----
dense_2 (Dense)         (None, 39)        20007
=====
Total params: 15,784,807
Trainable params: 1,070,119
Non-trainable params: 14,714,688
-----
```

```
[13]: # les couches de convolution précédemment freezed sont partiellement unfreezed :
for layer in base_model.layers[-4:]:
    layer.trainable = True

# model.load_weights(weights_path)

# le cnn est compilé
model.compile(optimizer=Adam(lr=1e-4), loss='categorical_crossentropy',
               metrics=['accuracy'])

# la structure du modèle est affichée
model.summary()

# le CNN est entraîné
for i in range(0, n_epochs):
    training(model, 1)
```

Model: "sequential"

| Layer (type) | Output Shape | Param # |
|------------------------------|-------------------------|----------|
| vgg16 (Functional) | (None, None, None, 512) | 14714688 |
| global_average_pooling2d (G1 | (None, 512) | 0 |
| dense (Dense) | (None, 1024) | 525312 |
| dropout (Dropout) | (None, 1024) | 0 |
| dense_1 (Dense) | (None, 512) | 524800 |
| dropout_1 (Dropout) | (None, 512) | 0 |

```
dense_2 (Dense)           (None, 39)          20007
=====
Total params: 15,784,807
Trainable params: 8,149,543
Non-trainable params: 7,635,264

[14]: # le cnn est compilé
model.compile(optimizer=Adam(lr=1e-5), loss='categorical_crossentropy', metrics=['accuracy'])

# la structure du modèle est affichée
model.summary()

# le CNN est entraîné
for i in range(0, n_epochs):
    training(model, 1)

Model: "sequential"

Layer (type)             Output Shape        Param #
=====
vgg16 (Functional)       (None, None, None, 512) 14714688
global_average_pooling2d (Gl (None, 512)      0
dense (Dense)            (None, 1024)         525312
dropout (Dropout)        (None, 1024)         0
dense_1 (Dense)          (None, 512)          524800
dropout_1 (Dropout)      (None, 512)          0
dense_2 (Dense)          (None, 39)           20007
=====

Total params: 15,784,807
Trainable params: 8,149,543
Non-trainable params: 7,635,264
```

3 Dataviz des résultats

```
[15]: import pandas as pd
import numpy as np

history = pickle.load(open(history_path, 'rb'))
```

```

df_history = pd.DataFrame()

for key, value in history.items():
    df_history[key] = value

n_last_history_values = 10

# description mathématique des fonctions coûts et des scores du CNN : pour
# derniers entraînements
print('Description mathématique des ', str(len(df_history[-n_last_history_values:])), ' derniers epochs')
df_history[-n_last_history_values:].describe()

```

Description mathématique des 10 derniers epochs

| | accuracy | val_accuracy | loss | val_loss |
|-------|-----------|--------------|-----------|-----------|
| count | 10.000000 | 10.000000 | 10.000000 | 10.000000 |
| mean | 0.938711 | 0.994259 | 0.200383 | 0.019787 |
| std | 0.002027 | 0.000558 | 0.006654 | 0.002225 |
| min | 0.935814 | 0.992976 | 0.191963 | 0.017291 |
| 25% | 0.937221 | 0.994054 | 0.194651 | 0.018263 |
| 50% | 0.938607 | 0.994303 | 0.201331 | 0.019642 |
| 75% | 0.940291 | 0.994511 | 0.203765 | 0.020415 |
| max | 0.941442 | 0.995077 | 0.210139 | 0.025044 |

```

[16]: import matplotlib.pyplot as plt
%matplotlib inline
plt.style.use("ggplot")

medianprops = {'color':'black'}
meanprops = {'marker':'o', 'markeredgecolor':'black',
            'markerfacecolor':'firebrick'}

plt.figure(figsize=(14, 4))
plt.subplot(121)
plt.boxplot([df_history[-n_last_history_values:].accuracy,
            df_history[-n_last_history_values:].val_accuracy],
            labels=["accuracy", "val_accuracy"],
            showfliers=False,
            medianprops=medianprops,
            vert=True,
            patch_artist=True,
            showmeans=True,
            meanprops=meanprops)

```

```

plt.title('Distribution : ' + str(len(df_history[-n_last_history_values:])) + ' derniers epochs')
plt.xlabel('Score')
plt.ylabel('Pourcentage (x 0.01)')
# plt.ylim(0.96, 1.0)

plt.subplot(122)
# Courbe de la précision sur l'échantillon d'entraînement
plt.plot(np.arange(len(df_history) - (n_last_history_values - 1), len(df_history) + 1, 1),
         df_history[-n_last_history_values:].accuracy,
         label = 'Training Accuracy',
         color = 'blue')

# Courbe de la précision sur l'échantillon de validation
plt.plot(np.arange(len(df_history) - (n_last_history_values - 1), len(df_history) + 1, 1),
         df_history[-n_last_history_values:].val_accuracy,
         label = 'Validation Accuracy',
         color = 'red')

# Labels des axes
plt.xlabel('Epochs')
plt.ylabel('Accuracy')

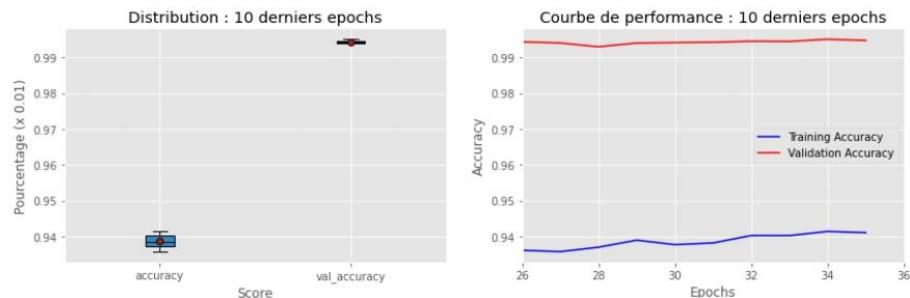
plt.title('Courbe de performance : ' + str(n_last_history_values) + ' derniers epochs')

plt.xlim(len(df_history) - (n_last_history_values - 1), len(df_history) + 1)
# plt.ylim(0.96, 1.0)

# Affichage de la légende
plt.legend()

# Affichage de la figure
plt.show();

```



3.1 Compte rendu de l'entraînement du CNN

3.1.1 10 premières epochs :

- Entrainement avec les couches de la partie convolution figées

3.1.2 10 epochs suivantes :

- Entrainement avec les 4 dernières couches de la partie convolution non figées
- Ajustement du learning rate avec la valeur = 10e-4

3.1.3 15 epochs suivantes :

- Entrainement avec les 4 dernières couches de la partie convolution non figées
- Ajustement du learning rate avec la valeur = 10e-5

```
[17]: plt.figure(figsize=(14, 4))
plt.subplot(121)

# Courbe de la perte sur l'échantillon d'entraînement
plt.plot(np.arange(1, len(df_history['loss']) + 1, 1),
         df_history.loss,
         label = 'Training Loss',
         color = 'blue')

# Courbe de la perte sur l'échantillon de validation
plt.plot(np.arange(1, len(df_history['val_loss']) + 1, 1),
         df_history.val_loss,
         label = 'Validation Loss',
         color = 'red')

# Labels des axes
plt.xlabel('Epochs')
plt.ylabel('Loss')

# Affichage de la légende
plt.legend()

# affichage du titre
plt.title('Courbe de perte')

plt.subplot(122)

# Courbe de la précision sur l'échantillon d'entraînement
plt.plot(np.arange(1, len(df_history['accuracy']) + 1, 1),
         df_history.accuracy,
```

```

    label = 'Training Accuracy',
    color = 'blue')

# Courbe de la précision sur l'échantillon de validation
plt.plot(np.arange(1 , len(df_history['val_accuracy']) + 1, 1),
         df_history.val_accuracy,
         label = 'Validation Accuracy',
         color = 'red')

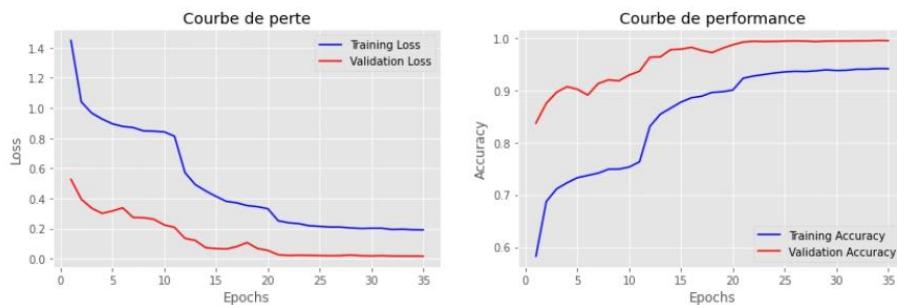
# Labels des axes
plt.xlabel('Epochs')
plt.ylabel('Accuracy')

# Affichage de la légende
plt.legend()

# affichage du titre
plt.title('Courbe de performance')

# Affichage de la figure
plt.show();

```



```

[18]: # Prédictions des classes : https://keras.io/api/applications/
       # Paragraphe : Extract features from an arbitrary intermediate layer with VGG19

from tensorflow.keras.preprocessing import image
import numpy as np

model.load_weights(weights_path)

indice_value_classes = {v: k for k, v in train_set.class_indices.items()}

# affiche le résultat des prédictions

```

```

def predictions(files_to_predict):

    # Prédire la classe de chaque fichier
    for file in listdirectory(dataset_test):

        img = image.load_img(file, target_size=(image_size, image_size))
        img = image.img_to_array(img)
        img = np.expand_dims(img, axis=0)
        img = preprocess_input(img)

        predict = model.predict(img)
        classe_indices = train_set.class_indices
        indice_predict = np.argmax(predict)
        trust_rate = predict[0][indice_predict]

        file_name = os.path.split(file)[1]

        result = '\n\nFichier : \t\t' + file_name + \
                 '\nClasse prédictée : \t' + indice_value_classes[indice_predict] + \
                 '\nIndice de confiance : \t' + str(round(trust_rate * 100, 2)) + \
                 '%'

        print(result)

```

```

[19]: # Récupérer la liste des fichiers à prédire
      from lib.tools import listdirectory

      # le jeu de données de test sert à évaluer les prédictions du CNN
      predictions(listdirectory(dataset_test))

```

Fichier : AppleCedarRust1.JPG

Classe prédictée : Apple___Cedar_apple_rust

Indice de confiance : 100.0%

Fichier : AppleCedarRust2.JPG

Classe prédictée : Apple___Cedar_apple_rust

Indice de confiance : 100.0%

Fichier : AppleCedarRust3.JPG

Classe prédictée : Apple___Cedar_apple_rust

Indice de confiance : 100.0%

Fichier : AppleCedarRust4.JPG
Classe prédictive : Apple___Cedar_apple_rust
Indice de confiance : 100.0%

Fichier : AppleScab1.JPG
Classe prédictive : Apple___Apple_scab
Indice de confiance : 100.0%

Fichier : AppleScab2.JPG
Classe prédictive : Apple___Apple_scab
Indice de confiance : 100.0%

Fichier : AppleScab3.JPG
Classe prédictive : Apple___Apple_scab
Indice de confiance : 100.0%

Fichier : CornCommonRust1.JPG
Classe prédictive : Corn_(maize)___Common_rust_
Indice de confiance : 100.0%

Fichier : CornCommonRust2.JPG
Classe prédictive : Corn_(maize)___Common_rust_
Indice de confiance : 100.0%

Fichier : CornCommonRust3.JPG
Classe prédictive : Corn_(maize)___Common_rust_
Indice de confiance : 100.0%

Fichier : PotatoEarlyBlight1.JPG
Classe prédictive : Potato___Early_blight
Indice de confiance : 100.0%

Fichier : PotatoEarlyBlight2.JPG
Classe prédictive : Potato___Early_blight
Indice de confiance : 100.0%

Fichier : PotatoEarlyBlight3.JPG
Classe prédictive : Potato___Early_blight
Indice de confiance : 100.0%

Fichier : PotatoEarlyBlight4.JPG
Classe prédictive : Potato___Early_blight
Indice de confiance : 100.0%

Fichier : PotatoEarlyBlight5.JPG
Classe prédictive : Potato___Early_blight
Indice de confiance : 100.0%

Fichier : PotatoHealthy1.JPG
Classe prédictive : Potato___healthy
Indice de confiance : 100.0%

Fichier : PotatoHealthy2.JPG
Classe prédictive : Potato___healthy
Indice de confiance : 100.0%

Fichier : TomatoEarlyBlight1.JPG
Classe prédictive : Tomato___Early_blight
Indice de confiance : 96.68%

Fichier : TomatoEarlyBlight2.JPG
Classe prédictive : Tomato___Early_blight
Indice de confiance : 100.0%

Fichier : TomatoEarlyBlight3.JPG
Classe prédictive : Tomato___Early_blight
Indice de confiance : 99.66%

Fichier : TomatoEarlyBlight4.JPG
Classe prédictive : Tomato___Early_blight
Indice de confiance : 100.0%

Fichier : TomatoEarlyBlight5.JPG
Classe prédictive : Tomato___Early_blight
Indice de confiance : 100.0%

Fichier : TomatoEarlyBlight6.JPG

Classe prédictive : Tomato___Early_blight
Indice de confiance : 91.92%

Fichier : TomatoHealthy1.JPG
Classe prédictive : Tomato___healthy
Indice de confiance : 100.0%

Fichier : TomatoHealthy2.JPG
Classe prédictive : Tomato___healthy
Indice de confiance : 100.0%

Fichier : TomatoHealthy3.JPG
Classe prédictive : Tomato___healthy
Indice de confiance : 100.0%

Fichier : TomatoHealthy4.JPG
Classe prédictive : Tomato___healthy
Indice de confiance : 100.0%

Fichier : TomatoYellowCurlVirus1.JPG
Classe prédictive : Tomato___Tomato_Yellow_Leaf_Curl_Virus
Indice de confiance : 100.0%

Fichier : TomatoYellowCurlVirus2.JPG
Classe prédictive : Tomato___Tomato_Yellow_Leaf_Curl_Virus
Indice de confiance : 100.0%

Fichier : TomatoYellowCurlVirus3.JPG
Classe prédictive : Tomato___Tomato_Yellow_Leaf_Curl_Virus
Indice de confiance : 100.0%

Fichier : TomatoYellowCurlVirus4.JPG
Classe prédictive : Tomato___Tomato_Yellow_Leaf_Curl_Virus
Indice de confiance : 100.0%

Fichier : TomatoYellowCurlVirus5.JPG
Classe prédictive : Tomato___Tomato_Yellow_Leaf_Curl_Virus
Indice de confiance : 100.0%

Fichier : TomatoYellowCurlVirus6.JPG
Classe prédictive : Tomato___Tomato_Yellow_Leaf_Curl_Virus
Indice de confiance : 100.0%

Fichier : _chat.jpg
Classe prédictive : Other
Indice de confiance : 100.0%

Fichier : _visage_femme.jpg
Classe prédictive : Other
Indice de confiance : 100.0%

10.4. Segmentation

Segmentation_d_image_Unet

December 25, 2020

1 Segmentation des images de feuilles

2 par un modèle de Deep Learning type U-Net

```
[ ]: ### Import du google drive pour l'enregistrement du modèle ou des poids
from google.colab import drive
drive.mount('/content/drive')
```

Mounted at /content/drive

2.1 Principe de la segmentation

La segmentation consiste à déterminer où se trouve l'objet d'intérêt dans l'image et de supprimer l'arrière plan.

Une méthode possible est l'utilisation d'un masque. Un masque est une image contenant seulement deux valeurs pour chaque pixel : 0 ou 1. Un pixel avec la valeur 1 est un pixel de l'objet à segmenter. Sinon le pixel correspond à l'arrière-plan et prend la valeur 0. On obtient ainsi une image en noir et blanc avec en blanc l'espace correspondant à l'objet.

Le modèle développé va prendre en entrée une image de feuille et renverra un masque de cette feuille.

2.2 Instanciation du modèle

Premièrement, il est nécessaire de créer l'architecture du modèle.

```
[ ]: ### Import des modules nécessaires pour l'ensemble du travail

import numpy as np
import os
import glob
import cv2
from tensorflow.keras import callbacks
import pandas as pd
import skimage.io as io
import skimage.transform as trans
import numpy as np
import tensorflow as tf
```

```

import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from tensorflow.keras.models import *
from tensorflow.keras.layers import *
from tensorflow.keras.optimizers import *
from tensorflow.keras.callbacks import ModelCheckpoint, LearningRateScheduler
from tensorflow.keras import backend as keras

[ ]: ### Définition d'une fonction de perte : Coefficient de Dice
def LossDice(y_true, y_pred):
    numerateur = tf.reduce_sum(y_true*y_pred, axis=(1, 2))
    denominateur= tf.reduce_sum(y_true+y_pred, axis=(1, 2))
    dice=2*numerateur/(denominateur+1E-4)
    return 1-dice

[ ]: ### Définition de la structure du modèle avec ses différentes couches

def unet(pretrained_weights = None, input_size = (256,256,3)):
    inputs = Input(input_size)
    conv1 = Conv2D(32, 3, activation='relu', padding='same')(inputs)
    conv1 = Conv2D(32, 3, activation='relu', padding='same')(conv1)
    pool1 = MaxPooling2D(pool_size=2)(conv1)

    conv2 = Conv2D(64, 3, activation='relu', padding='same')(pool1)
    conv2 = Conv2D(64, 3, activation='relu', padding='same')(conv2)
    pool2 = MaxPooling2D(pool_size=(2))(conv2)

    conv3 = Conv2D(128, 3, activation='relu', padding='same')(pool2)
    conv3 = Conv2D(128, 3, activation='relu', padding='same')(conv3)
    pool3 = MaxPooling2D(pool_size=(2))(conv3)

    conv4 = Conv2D(256, 3, activation='relu', padding='same')(pool3)
    conv4 = Conv2D(256, 3, activation='relu', padding='same')(conv4)
    pool4 = MaxPooling2D(pool_size=(2))(conv4)

    conv5 = Conv2D(512, 3, activation='relu', padding='same')(pool4)
    conv5 = Conv2D(512, 3, activation='relu', padding='same')(conv5)

    up6 = concatenate([UpSampling2D(size=(2))(conv5), conv4], axis=-1)
    conv6 = Conv2D(256, 3, activation='relu', padding='same')(up6)
    conv6 = Conv2D(256, 3, activation='relu', padding='same')(conv6)

    up7 = concatenate([UpSampling2D(size=(2))(conv6), conv3], axis=-1)
    conv7 = Conv2D(128, 3, activation='relu', padding='same')(up7)
    conv7 = Conv2D(128, 3, activation='relu', padding='same')(conv7)

    up8 = concatenate([UpSampling2D(size=(2))(conv7), conv2], axis=-1)

```

```

conv8 = Conv2D(64, 3, activation='relu', padding='same')(up8)
conv8 = Conv2D(64, 3, activation='relu', padding='same')(conv8)

up9 = concatenate([UpSampling2D(size=(2))(conv8), conv1], axis=-1)
conv9 = Conv2D(32, (3), activation='relu', padding='same')(up9)
conv9 = Conv2D(32, (3), activation='relu', padding='same')(conv9)

conv10 = Conv2D(1, 1, activation='sigmoid')(conv9)

### Compilation du modèle avec la fonction de perte 'loss dice' et la
# métrique 'accuracy'
model = Model(inputs=inputs, outputs=conv10)
model.compile(optimizer=Adam(lr=1e-3), loss=LossDice,
              metrics=['accuracy'])

model.summary()

### Si le modèle a précédemment été entrainé, les poids sauvegardés sont
# chargés
if (pretrained_weights):
    model.load_weights(pretrained_weights)
if (pretrained_weights):
    print("\nLe modèle a été instancié avec les poids chargés.")

return model

```

[]: *Chargement de poids sauvegardés s'ils existent*

```

export_dir='/content/drive/MyDrive/'
pretrained_weights = None
if os.path.exists(export_dir+'weights.h5'):
    pretrained_weights = export_dir+'weights.h5'

Instanciation d'un modèle
unet=unet(input_size=(256,256,3), pretrained_weights=pretrained_weights)

```

Model: "model"

| Layer (type) | Output Shape | Param # | Connected to |
|----------------------|--------------------------|---------|----------------|
| input_1 (InputLayer) | [(None, 256, 256, 3) 0 | | |
| conv2d (Conv2D) | (None, 256, 256, 32) 896 | | input_1[0] [0] |

| | | | |
|--------------------------------|----------------------|---------|-----------------|
| conv2d_1 (Conv2D) | (None, 256, 256, 32) | 9248 | conv2d[0] [0] |
| max_pooling2d (MaxPooling2D) | (None, 128, 128, 32) | 0 | conv2d_1[0] [0] |
| conv2d_2 (Conv2D) | (None, 128, 128, 64) | 18496 | conv2d_1[0] [0] |
| max_pooling2d[0] [0] | | | |
| conv2d_3 (Conv2D) | (None, 128, 128, 64) | 36928 | conv2d_2[0] [0] |
| max_pooling2d_1 (MaxPooling2D) | (None, 64, 64, 64) | 0 | conv2d_3[0] [0] |
| conv2d_4 (Conv2D) | (None, 64, 64, 128) | 73856 | conv2d_3[0] [0] |
| max_pooling2d_1[0] [0] | | | |
| conv2d_5 (Conv2D) | (None, 64, 64, 128) | 147584 | conv2d_4[0] [0] |
| max_pooling2d_2 (MaxPooling2D) | (None, 32, 32, 128) | 0 | conv2d_5[0] [0] |
| conv2d_6 (Conv2D) | (None, 32, 32, 256) | 295168 | conv2d_5[0] [0] |
| max_pooling2d_2[0] [0] | | | |
| conv2d_7 (Conv2D) | (None, 32, 32, 256) | 590080 | conv2d_6[0] [0] |
| max_pooling2d_3 (MaxPooling2D) | (None, 16, 16, 256) | 0 | conv2d_7[0] [0] |
| conv2d_8 (Conv2D) | (None, 16, 16, 512) | 1180160 | conv2d_7[0] [0] |
| max_pooling2d_3[0] [0] | | | |
| conv2d_9 (Conv2D) | (None, 16, 16, 512) | 2359808 | conv2d_8[0] [0] |
| up_sampling2d (UpSampling2D) | (None, 32, 32, 512) | 0 | conv2d_9[0] [0] |
| concatenate (Concatenate) | (None, 32, 32, 768) | 0 | |

```
up_sampling2d[0] [0]                                     conv2d_7[0] [0]
-----
conv2d_10 (Conv2D)          (None, 32, 32, 256) 1769728
concatenate[0] [0]
-----
conv2d_11 (Conv2D)          (None, 32, 32, 256) 590080    conv2d_10[0] [0]
-----
up_sampling2d_1 (UpSampling2D) (None, 64, 64, 256) 0      conv2d_11[0] [0]
-----
concatenate_1 (Concatenate) (None, 64, 64, 384) 0
up_sampling2d_1[0] [0]                                     conv2d_5[0] [0]
-----
conv2d_12 (Conv2D)          (None, 64, 64, 128) 442496
concatenate_1[0] [0]
-----
conv2d_13 (Conv2D)          (None, 64, 64, 128) 147584    conv2d_12[0] [0]
-----
up_sampling2d_2 (UpSampling2D) (None, 128, 128, 128) 0      conv2d_13[0] [0]
-----
concatenate_2 (Concatenate) (None, 128, 128, 192) 0
up_sampling2d_2[0] [0]                                     conv2d_3[0] [0]
-----
conv2d_14 (Conv2D)          (None, 128, 128, 64) 110656
concatenate_2[0] [0]
-----
conv2d_15 (Conv2D)          (None, 128, 128, 64) 36928    conv2d_14[0] [0]
-----
up_sampling2d_3 (UpSampling2D) (None, 256, 256, 64) 0      conv2d_15[0] [0]
-----
concatenate_3 (Concatenate) (None, 256, 256, 96) 0
up_sampling2d_3[0] [0]                                     conv2d_1[0] [0]
```

```

-----
conv2d_16 (Conv2D)           (None, 256, 256, 32) 27680
concatenate_3[0] [0]

-----
conv2d_17 (Conv2D)           (None, 256, 256, 32) 9248      conv2d_16[0] [0]

-----
conv2d_18 (Conv2D)           (None, 256, 256, 1)   33       conv2d_17[0] [0]
=====
=====
Total params: 7,846,657
Trainable params: 7,846,657
Non-trainable params: 0
=====

-----
Le modèle a été instancié avec les poids chargés.

```

2.3 Importation et mis en forme des données

Les données sont récupérées sur kaggle. Les images segmentées présentes dans ce jeu de données sont transformées en masque.

```

[ ]: ### L'importation des données se fait directement par kaggle
! pip install -q kaggle
from google.colab import files
files.upload()

<IPython.core.display.HTML object>
Saving kaggle.json to kaggle.json

[ ]: {'kaggle.json':
      b'{"username":"ralaivaomathis","key":"638bba9daf4d63aeeec6be9b4a3d0ac7a"}'

[2]: ! mkdir ~/.kaggle
! cp kaggle.json ~/.kaggle/
! chmod 600 ~/.kaggle/kaggle.json
! kaggle datasets download -d abdallahhalidev/plantvillage-dataset
### Les données seront stockées dans un dossier 'dataset'
! mkdir dataset
! unzip plantvillage-dataset.zip -d dataset

```

Dans la partie suivante, nous allons convertir les images segmentées du jeu de données en masques.

En effet, le modèle est plus performant à prédire des masques que des images segmentées directement.

```
[ ]: ### Génération de masques à partir des images segmentées du jeu de données

### Premièrement, on récupère les chemins des dossiers contenant les images
path_folders=glob.glob('/content/dataset/plantvillage dataset/segmented/**')

### Affichage d'un exemple d'image segmentée

path_example= glob.glob(path_folders[0]+ '**')[0]
plt.figure(figsize=(15,7))
plt.subplot(121)
example=cv2.imread(path_example)
plt.imshow(example)
plt.title("Image segmentée d'origine")
plt.axis('off')

### Transformation des images segmentées en masques

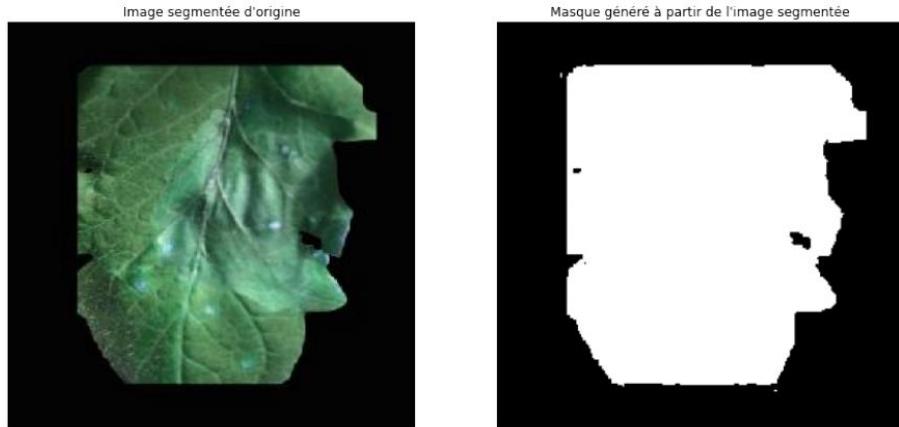
threshold=20
for path in path_folders:
    for fichier in glob.glob(path+'/**'):
        img=cv2.imread(fichier)
        if len(img.shape)==3:

            ### La moyenne des pixels RGB est calculée et un seuil est défini pour séparer les pixels
            ### de la feuille et le reste.
            ### La valeur de 1 est attribuée pour un pixel de la feuille, le pixel prend la valeur 0 sinon.

            img_mask=(img.mean(axis=2)>threshold).astype(int)
            cv2.imwrite(fichier,img_mask)

### Exemple de masque produit

plt.subplot(122)
example=cv2.imread(path_example, cv2.IMREAD_GRAYSCALE)
plt.imshow(example*255,cmap='gray')
plt.title("Masque généré à partir de l'image segmentée")
plt.axis('off');
```



```
[ ]: #### Crédation d'un datafame contenant les chemins des images et des masques
      ↳ correspondants

#### On récupère les chemins des masques et des images

directory_images='/content/dataset/plantvillage dataset/color'
directory_masks='/content/dataset/plantvillage dataset/segmented'
path_masks=[]
path_images=[]

for path in glob.glob(directory_masks+'/*'):
    path_masks+=glob.glob(path+'/*')

for path in glob.glob(directory_images+'/*'):
    path_images+=glob.glob(path+'/*')

## Les listes de chemins des masks et images sont triées
## pour s'assurer de leur correspondance

path_masks=sorted(path_masks)
path_images=sorted(path_images)

## Suppression d'un mask sans image correspondante

indice=12292
print(path_masks[indice],path_images[indice])
del(path_masks[indice])

## Crédation du DataFame
```

```
dico={'Image':path_images,'Mask':path_masks}
df=pd.DataFrame(dico)

/content/dataset/plantvillage dataset/segmented/Grape__Esca_(Black_Measles)/7e1
fd9b9-1fd9-4f98-93f5-8cf9ebc60dd9__FAM_B.Msls 4430_final_masked.jpg
/content/dataset/plantvillage dataset/color/Grape__Esca_(Black_Measles)/7e4cf44
f-02ec-4812-9afa-b19f45add835__FAM_B.Msls 4422.JPG
```

Les jeux de données d'entraînement, de validation et de test sont générés à partir du DataFrame créé. Les données seront chargées au fur et à mesure de l'entraînement du modèle par une fonction définie ci-dessous.

Pour l'entraînement du modèle, les features correspondent aux images et les valeurs à prédire (les labels) sont les masques correspondants.

```
[ ]: ### Définition de fonctions de chargement et transformation
### des images ou masques à partir des chemins

import tensorflow as tf
def load_image(filepath,resize=(256,256)):
    # Charger l'information brute en mémoire
    im = tf.io.read_file(filepath)
    # Décoder l'information en un tensorflow RGB (3 channels).
    im = tf.io.decode_jpeg(im, channels=3)
    #Redimensionner l'image
    return tf.image.resize(im, size=resize)

def load_mask(filepath,resize=(256,256)):
    im = tf.io.read_file(filepath)
    ### Dans le cas des masques, l'image est en noir et blanc, il n'y a donc
↪ qu'une valeur par pixel
    im = tf.io.decode_jpeg(im, channels=1)
    return tf.image.resize(im, size=resize)
```

```
[ ]: ### Crédit du jeu d'entraînement, de validation et de test
### La fonction définie au dessus est utilisée pour charger les images et
### masques au fur et à mesure de l'entraînement

X_train_path, X_test_path, y_train_path, y_test_path = train_test_split(df.
    ↪ Image, df.Mask, test_size=0.2, random_state=456)
X_train_path, X_val_path, y_train_path, y_val_path = ↪
    ↪ train_test_split(X_train_path, y_train_path, test_size=0.2, random_state=456)

dataset_train = tf.data.Dataset.from_tensor_slices((X_train_path,y_train_path))
dataset_train = dataset_train.map(lambda x, y : [load_image(x),load_mask(y)], ↪
    ↪ num_parallel_calls=-1).batch(1)
```

```
dataset_val = tf.data.Dataset.from_tensor_slices((X_val_path,y_val_path))
dataset_val = dataset_val.map(lambda x, y : [load_image(x),load_mask(y)],  

    num_parallel_calls=-1).batch(1)

dataset_test = tf.data.Dataset.from_tensor_slices((X_test_path,y_test_path))
dataset_test = dataset_test.map(lambda x, y : [load_image(x),load_mask(y)],  

    num_parallel_calls=-1).batch(1)
```

[]: *### Définition de callbacks et entrainement du modèle*

```
### Sauvegarde automatique des poids
checkpoint = callbacks.ModelCheckpoint(filepath = '/content/drive/MyDrive/  

    checkpoint/weights.h5',  

        monitor = 'loss',  

        save_best_only = True,  

        save_weights_only = True,  

        mode = 'min',  

        save_freq = 'epoch')

### 'Callback' permettant d'ajuster le taux d'apprentissage au cours de  

    l'entraînement
lr_plateau = callbacks.ReduceLROnPlateau(monitor = 'loss',  

        patience=1,  

        factor=0.1,  

        verbose=2,  

        mode='min')
```

training=unet.fit(dataset_train, epochs=3, validation_data=dataset_val,
 callbacks = [lr_plateau, checkpoint])

```
Epoch 1/3
34755/34755 [=====] - 1048s 30ms/step - loss: 0.0610 -
accuracy: 0.9497 - val_loss: 0.0690 - val_accuracy: 0.9416
WARNING:tensorflow:Can save best model only with <function LossDice at
0x7f15671adbf8> available, skipping.
Epoch 2/3
34755/34755 [=====] - 1037s 30ms/step - loss: 0.0613 -
accuracy: 0.9495 - val_loss: 0.0649 - val_accuracy: 0.9460
WARNING:tensorflow:Can save best model only with <function LossDice at
0x7f15671adbf8> available, skipping.
Epoch 3/3
34755/34755 [=====] - 1034s 30ms/step - loss: 0.0615 -
accuracy: 0.9492 - val_loss: 0.0644 - val_accuracy: 0.9460

Epoch 00003: ReduceLROnPlateau reducing learning rate to 0.0001000000474974513.
WARNING:tensorflow:Can save best model only with <function LossDice at
0x7f15671adbf8> available, skipping.
```

```
[ ]: ### Sauvegarde des poids et du modèle entraîné
unet.save_weights(export_dir+'weights.h5')
unet.save(export_dir+'model.h5')
```

2.4 Evaluation de la précision du modèle et réalisation d'essais sur de nouvelles images

Le modèle est évalué sur le jeu de donnée de test et des images d'un autre jeu de données vont être chargées pour pouvoir tester la segmentation générée.

```
[ ]: ### Evaluation de la précision du modèle sur le jeu de test
evaluation=unet.evaluate(dataset_test)
print('Précision du modèle sur le jeu de test :',np.round(evaluation[1],2),"\n",
      'Perte du modèle sur le jeu de test :',np.round(evaluation[0],2))
```

```
10861/10861 [=====] - 95s 9ms/step - loss: 0.0644 -
accuracy: 0.9452
Précision du modèle sur le jeu de test : 0.95
Perte du modèle sur le jeu de test : 0.06
```

```
[ ]: ### Démonstration d'une prédiction de masque en comparaison avec un masque
### fait à partir de l'image segmentée du jeu de donnée Test

size=2
indexes=np.random.choice(np.array(X_test_path.index),size=2)

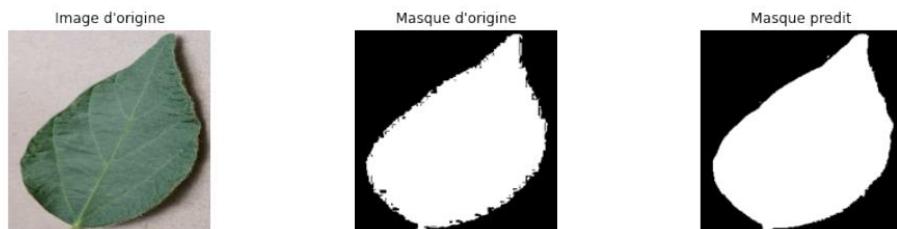
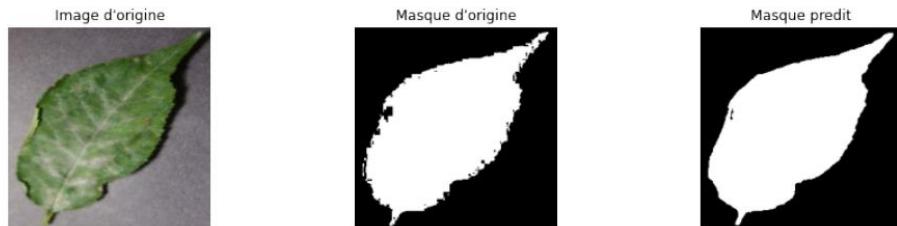
i=0
for idx in indexes:
    image=load_image(X_test_path[idx])
    img=tf.reshape(image,(1,256,256,3))
    mask_pred=unet.predict(img)
    mask_true=load_mask(y_test_path[idx])
    image=tf.cast(image,dtype=tf.int32)

    plt.figure(figsize=(15,7))

    i+=1
    plt.subplot(size,3,i)
    plt.imshow(image)
    plt.axis("off")
    plt.title("Image d'origine")

    i+=1
    plt.subplot(size,3,i)
    plt.imshow(tf.reshape(mask_true,(256,256)),cmap='gray')
    plt.axis("off")
    plt.title("Masque fait à partir de l'image segmentée")
```

```
i+=1
plt.subplot(size,3,i)
plt.imshow(tf.reshape(mask_pred,(256,256)),cmap='gray')
plt.axis("off")
plt.title("Masque predit par le modèle")
```



Une observation importante est mise en évidence par les images ci-dessus. Les masques générés par le modèle UNet semblent plus précis que ceux obtenus par transformation des images segmentées.

Cela explique le fait que le modèle n'arrive pas à converger à plus de 95% d'accuracy. Les masques sont bons mais ne correspondent pas tout à fait à ceux donnés en entraînement, il y a donc un écart qui fait chuter l'accuracy.

```
[1]: ### Importation d'un nouveau jeu de donnée pour voir la segmentation réalisée
! kaggle datasets download -d vipoooool/new-plant-diseases-dataset
### Les données seront stockés dans un dossier 'dataset_essais'
! mkdir dataset_essais
! unzip new-plant-diseases-dataset.zip -d dataset_essais
```

```
[ ]: ### On récupère les chemins des images à segmenter
directory_images='/content/dataset_essais/test/'
path_images=[]
for path in glob.glob(directory_images+'/*'):
    path_images+=glob.glob(path+'/*')
```

```
[ ]: ### Chargement éventuel d'un modèle enregistré
if os.path.exists(export_dir+'saved_model.h5'):
    saved_unet= tf.keras.models.load_model(export_dir+'saved_model.h5')

[ ]: ### Une image est choisie au hasard et la segmentation est appliquée
chiffre=np.random.choice(range(len(path_images)),size=1)[0]
im=cv2.imread(path_images[chiffre])
mask=load_image(path_images[chiffre])
mask=tf.reshape(mask,(1,256,256,3))
pred=unet.predict(mask)
#pred = saved_unet.predict(mask)

plt.figure(figsize=(15,7))
plt.subplot(221)
plt.imshow(im)
plt.title("Image d'origine")
plt.axis('off')

plt.subplot(222)
plt.imshow(pred.reshape(256,256),cmap='gray')
plt.title("Masque généré par le modèle")
plt.axis('off')

### La segmentation est réalisée en multipliant le masque et l'image à segmenter
seg=(im*pred.reshape(256,256,1)).astype(int)
plt.subplot(224)
plt.imshow(seg)
plt.title("Image segmentée")
plt.axis('off');

### Superposition du masque et de l'image
seg=(im*pred.reshape(256,256,1)).astype(int)
plt.subplot(223)
plt.imshow(im)
plt.imshow(pred.reshape(256,256),alpha=0.5)
plt.title("Mise en évidence de la feuille sur l'image")
plt.axis('off');
```



[]:

10.5. Identification d'objet et « bounding box »

bounding_box

December 25, 2020

```
[1]: import sys
import os
import glob
import cv2
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf

[2]: # initialisation
sys.path.append(os.path.abspath(os.path.join(os.getcwd(), os.pardir)))

from lib import ressources as res
racine = os.path.abspath(os.path.realpath(res.dir_root))

dataset_test = res.dir_dataset_test
print(dataset_test)
```

C:\Users\NOEL\dev\python\dataset\plant-dataset\test

```
[3]: ### Définition d'une fonction de perte : Coefficient de Dice
def LossDice(y_true, y_pred):
    numerateur = tf.reduce_sum(y_true*y_pred, axis=(1, 2))
    denominateur= tf.reduce_sum(y_true+y_pred, axis=(1, 2))
    dice=2*numerateur/(denominateur+1E-4)
    return 1-dice
```

```
[4]: # le modèle entraîné est récupéré
def load_saved_unet():

    model = tf.keras.models.load_model('data/unet_model.h5')
    model.summary()
    return model

unet = load_saved_unet()
```

Model: "functional_1"

| Layer (type) | Output Shape | Param # | Connected to |
|--------------|--------------|---------|--------------|
|--------------|--------------|---------|--------------|

```
=====
=====
input_1 (InputLayer)      [(None, 256, 256, 3) 0
-----
conv2d (Conv2D)          (None, 256, 256, 32) 896      input_1[0] [0]
-----
conv2d_1 (Conv2D)        (None, 256, 256, 32) 9248     conv2d[0] [0]
-----
max_pooling2d (MaxPooling2D) (None, 128, 128, 32) 0      conv2d_1[0] [0]
-----
conv2d_2 (Conv2D)        (None, 128, 128, 64) 18496
max_pooling2d[0] [0]
-----
conv2d_3 (Conv2D)        (None, 128, 128, 64) 36928     conv2d_2[0] [0]
-----
max_pooling2d_1 (MaxPooling2D) (None, 64, 64, 64) 0      conv2d_3[0] [0]
-----
conv2d_4 (Conv2D)        (None, 64, 64, 128) 73856
max_pooling2d_1[0] [0]
-----
conv2d_5 (Conv2D)        (None, 64, 64, 128) 147584     conv2d_4[0] [0]
-----
max_pooling2d_2 (MaxPooling2D) (None, 32, 32, 128) 0      conv2d_5[0] [0]
-----
conv2d_6 (Conv2D)        (None, 32, 32, 256) 295168
max_pooling2d_2[0] [0]
-----
conv2d_7 (Conv2D)        (None, 32, 32, 256) 590080     conv2d_6[0] [0]
-----
max_pooling2d_3 (MaxPooling2D) (None, 16, 16, 256) 0      conv2d_7[0] [0]
-----
conv2d_8 (Conv2D)        (None, 16, 16, 512) 1180160
max_pooling2d_3[0] [0]
-----
```

```
conv2d_9 (Conv2D)           (None, 16, 16, 512) 2359808      conv2d_8[0] [0]
-----
up_sampling2d (UpSampling2D) (None, 32, 32, 512) 0          conv2d_9[0] [0]
-----
concatenate (Concatenate)   (None, 32, 32, 768) 0
up_sampling2d[0] [0]
                                         conv2d_7[0] [0]
-----
conv2d_10 (Conv2D)           (None, 32, 32, 256) 1769728
concatenate[0] [0]
-----
conv2d_11 (Conv2D)           (None, 32, 32, 256) 590080      conv2d_10[0] [0]
-----
up_sampling2d_1 (UpSampling2D) (None, 64, 64, 256) 0          conv2d_11[0] [0]
-----
concatenate_1 (Concatenate)   (None, 64, 64, 384) 0
up_sampling2d_1[0] [0]
                                         conv2d_5[0] [0]
-----
conv2d_12 (Conv2D)           (None, 64, 64, 128) 442496
concatenate_1[0] [0]
-----
conv2d_13 (Conv2D)           (None, 64, 64, 128) 147584      conv2d_12[0] [0]
-----
up_sampling2d_2 (UpSampling2D) (None, 128, 128, 128) 0        conv2d_13[0] [0]
-----
concatenate_2 (Concatenate)   (None, 128, 128, 192) 0
up_sampling2d_2[0] [0]
                                         conv2d_3[0] [0]
-----
conv2d_14 (Conv2D)           (None, 128, 128, 64) 110656
concatenate_2[0] [0]
-----
conv2d_15 (Conv2D)           (None, 128, 128, 64) 36928       conv2d_14[0] [0]
```

```

up_sampling2d_3 (UpSampling2D)  (None, 256, 256, 64) 0           conv2d_15[0] [0]
-----
concatenate_3 (Concatenate)    (None, 256, 256, 96) 0
up_sampling2d_3[0] [0]          conv2d_1 [0] [0]
-----
conv2d_16 (Conv2D)            (None, 256, 256, 32) 27680
concatenate_3[0] [0]
-----
conv2d_17 (Conv2D)            (None, 256, 256, 32) 9248      conv2d_16[0] [0]
-----
conv2d_18 (Conv2D)            (None, 256, 256, 1)  33       conv2d_17[0] [0]
=====
=====
Total params: 7,846,657
Trainable params: 7,846,657
Non-trainable params: 0
=====
```

[5]: # chargement des images et masques après transformations

```

def load_image(filepath,resize=(256,256)):
    # l'information brute est chargée en mémoire
    im = tf.io.read_file(filepath)
    # l'information est décodée pour être transformée en un tensorflow RGB (3 channels)
    im = tf.io.decode_jpeg(im, channels=3)
    # l'image redimensionnée est retournée
    return tf.image.resize(im, size=resize)

def load_mask(filepath,resize=(256,256)):
    # l'information brute est chargée en mémoire
    im = tf.io.read_file(filepath)
    # dans le cas des masques, l'image est en noir et blanc, il n'y a donc qu'une valeur par pixel
    im = tf.io.decode_jpeg(im, channels=1)
    # retourne l'image redimensionnée
    return tf.image.resize(im, size=resize)
```

[6]: # les prédictions sont réalisées

```
def predictions():
```

```

images = []
masques = []

# chemin des images de test
path_images=[]
for path in glob.glob(dataset_test):
    path_images += glob.glob(path+'/*')

# pour chaque image
for path_image in path_images:
    # l'image est lue
    im = cv2.imread(path_image)
    # l'image est ajoutée
    images.append(im)

    # l'image est chargée avec transformations
    tmp_im = load_image(path_image)
    # l'image est redimensionnée
    tmp_im = tf.reshape(tmp_im,(1, 256, 256, 3))
    # prédiction du cnn qui renvoi le masque de l'image
    masque = unet.predict(tmp_im)
    # le masque retourné par le cnn est ajouté
    masques.append(masque)

    # suppression des variables
    del im
    del tmp_im
    del masque

# les images et les masques correspondants sont retournés
return images, masques

```

```

[7]: # les images issues de la prédiction du cnn sont ajoutées
def affiche_prediction(im, pred):

    # l'image d'origine est affichée
    plt.figure(figsize=(8, 4))
    plt.subplot(141)
    plt.imshow(im)
    plt.title("Image d'origine")
    plt.axis('off')

    # masque de la photo d'origine
    plt.subplot(142)
    plt.imshow(pred.reshape(256, 256),cmap='gray')
    plt.title("Masque")
    plt.axis('off')

```

```

# superposition du masque et de l'image
plt.subplot(143)
plt.imshow(im)
plt.imshow(pred.reshape(256,256), alpha=0.5)
plt.title("Superposition")
plt.axis('off');

# la segmentation est réalisée en multipliant le masque et l'image à
# segmenter
seg=(im * pred.reshape(256, 256, 1)).astype(int)
plt.subplot(144)
plt.imshow(seg)
plt.title("Segmentation")
plt.axis('off');
# cv2.imwrite('plante.png', seg)

# suppression des variables
del im
del pred
del seg

```

```

[8]: # pour chaque masque, les bounding box sont calculés et affichés avec l'image
      # correspondante
def bounding_box(image, mask):

    bounding_box_color=(255, 0, 0)
    countour_surface=10000

    # les dimensions inutiles sont supprimées
    mask = mask.squeeze()

    # le masque étant noir et blanc, il est redimensionné à 3 dimensions
    mask = cv2.merge([mask,mask,mask])

    # la normalisation est supprimée
    mask = mask * 256

    # les valeurs sont mises au format int8
    mask = mask.astype(np.uint8)

    # le masque au format cv2 est transformé en gris
    img = cv2.cvtColor(mask, cv2.COLOR_BGR2GRAY)

    # les pixels sont mis à 0 ou 1 suivant leur valeur par rapport au seuil
    # choisi

```

```

ret, thresh = cv2.threshold(src=img, thresh=127, maxval=255, type=0)

# les contours trouvés dans l'image sont détectés
contours, hierarchy = cv2.findContours(thresh, cv2.RETR_TREE, cv2.
                                         CHAIN_APPROX_SIMPLE)

# pour chaque contour trouvé
for contour in contours:

    # pour chaque contour donc la surface est suffisant pour être pris en
    ↪compte
    if cv2.contourArea(contour) > countour_surface:
        # print('Surface du contour -> ', cv2.contourArea(contour))
        x,y,w,h = cv2.boundingRect(contour)

        # l'image et le bounding box sont dessinés
        plt.figure(figsize=(5, 3))
        image = cv2.rectangle(img=image, pt1=(x, y), pt2=(x+w, y+h), ↪
        ↪color=bounding_box_color, thickness=2)
        plt.axis('off')
        plt.imshow(image)

cv2.destroyAllWindows()

# bounding_box(images[14], masques[14])

```

```

[9]: # prédictions du cnn
images, masques = predictions()

# pour chaque prédition
for i in range(0, len(images)):
    # les images issues de la prédition sont affichées
    affiche_prediction(images[i], masques[i])
    # les bounding box calculés à partir des masques sont affichés
    bounding_box(images[i], masques[i])

```

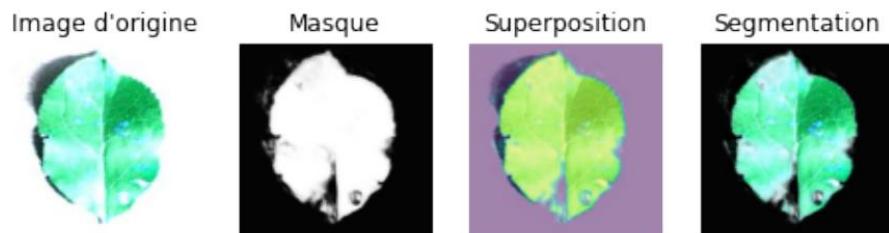




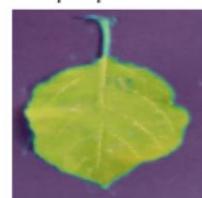
Image d'origine



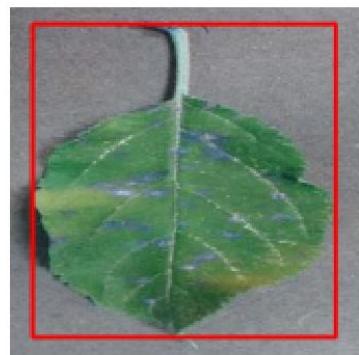
Masque



Superposition



Segmentation



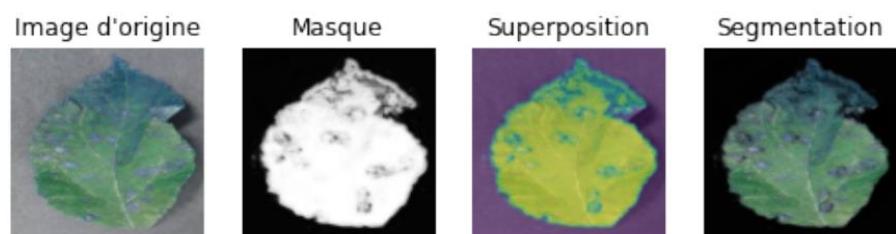
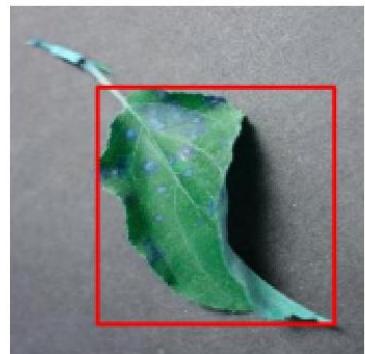
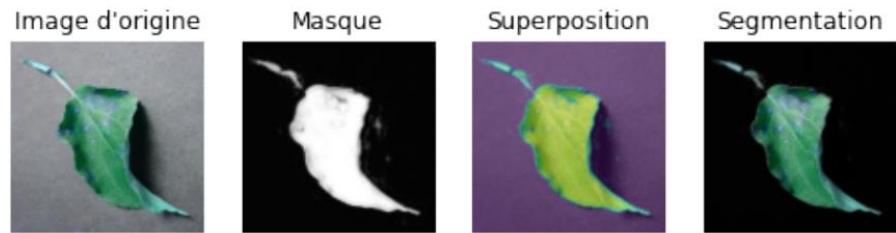




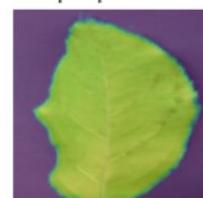
Image d'origine



Masque



Superposition



Segmentation



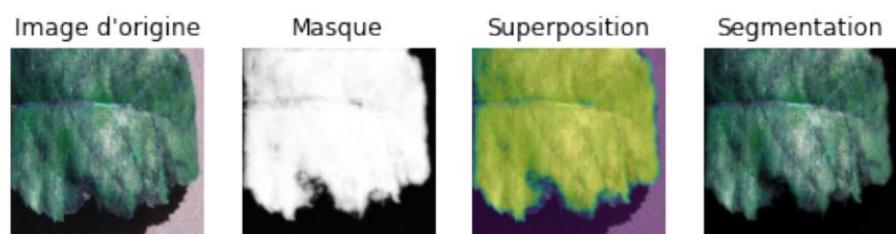
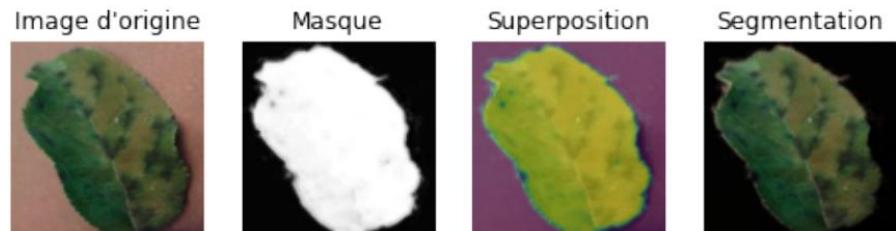




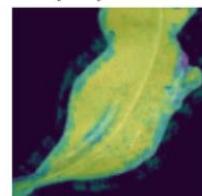
Image d'origine



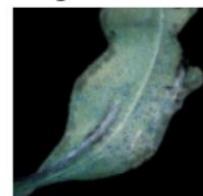
Masque



Superposition



Segmentation



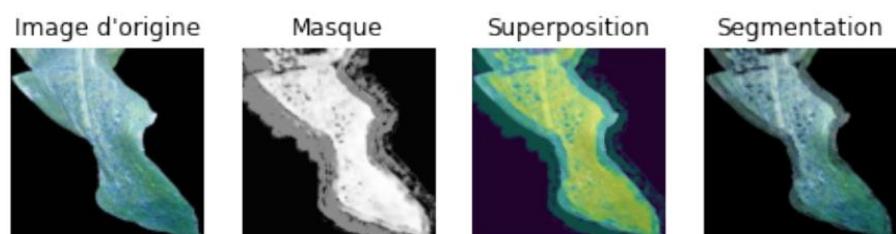
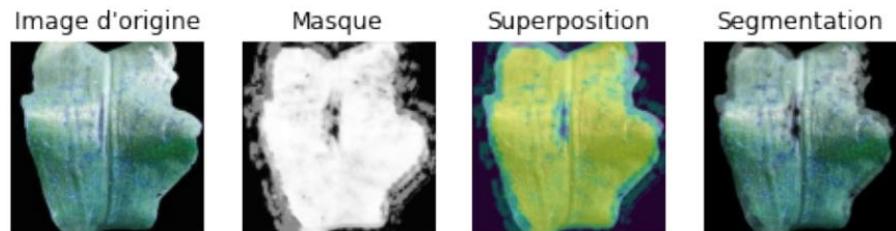
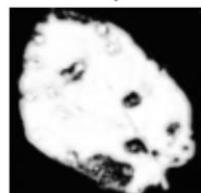




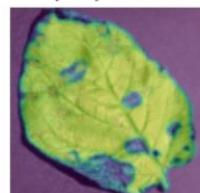
Image d'origine



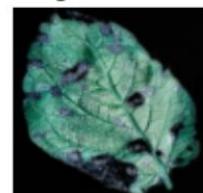
Masque



Superposition



Segmentation



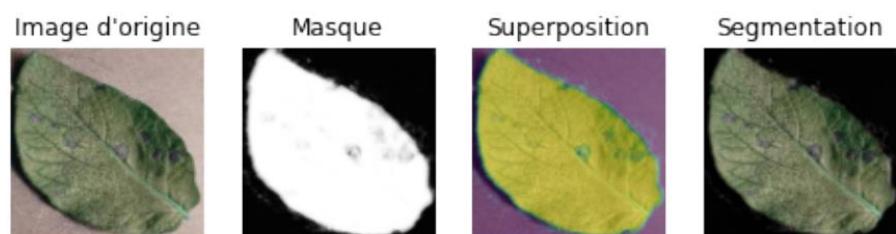
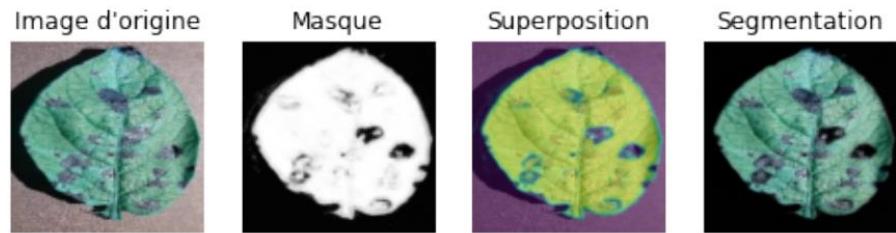
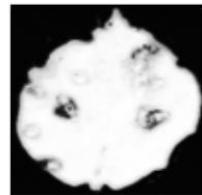




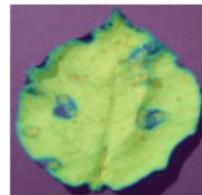
Image d'origine



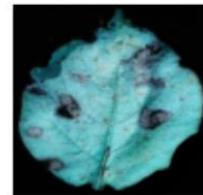
Masque

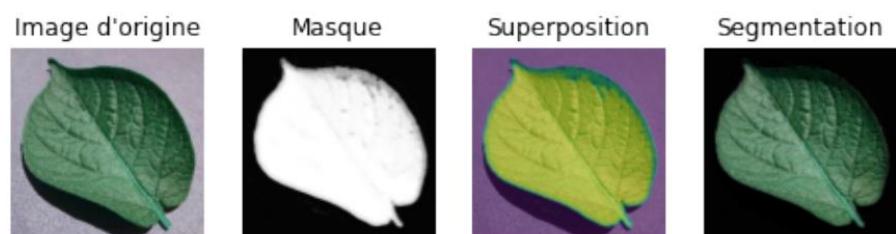
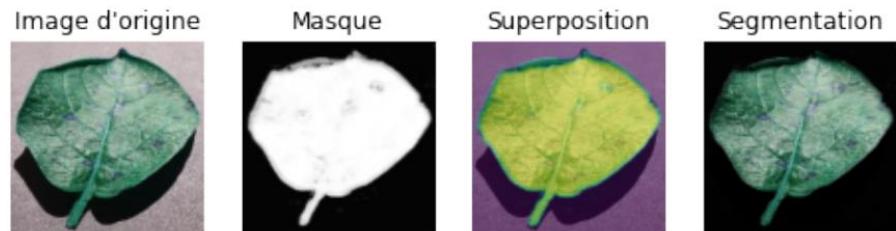


Superposition



Segmentation





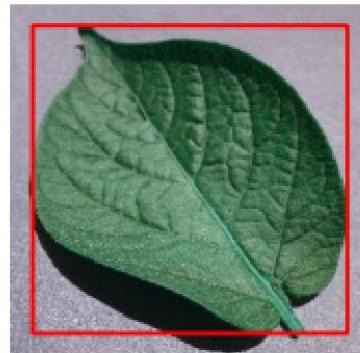
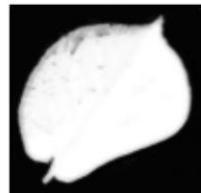


Image d'origine



Masque



Superposition



Segmentation



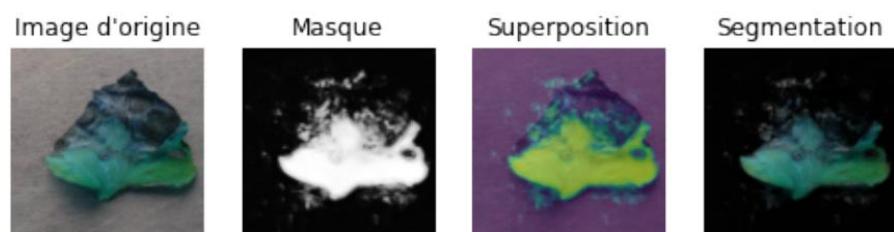
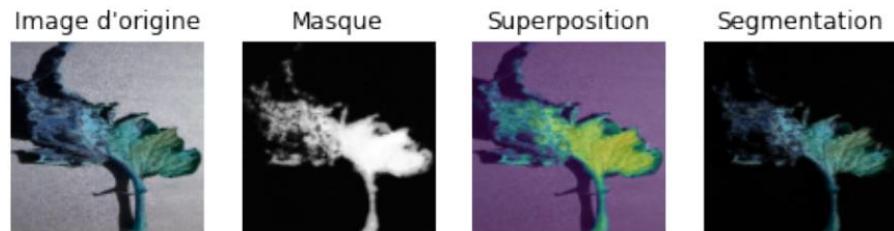
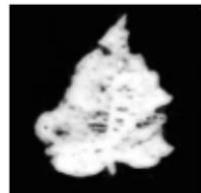




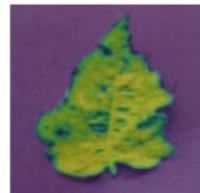
Image d'origine



Masque



Superposition



Segmentation



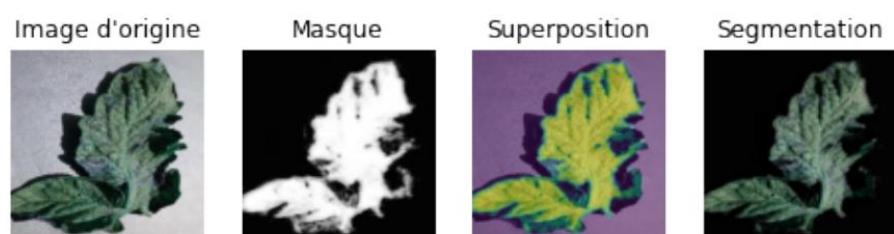
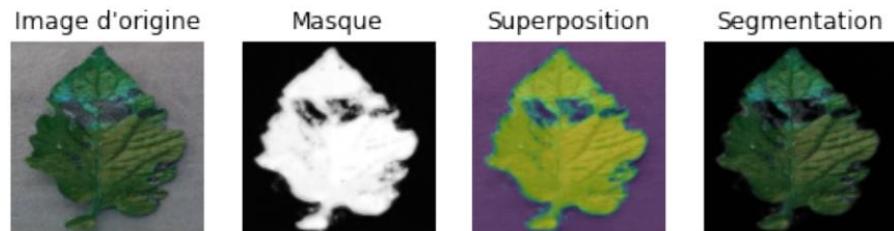
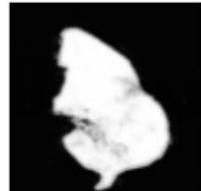




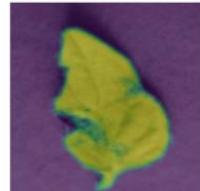
Image d'origine



Masque

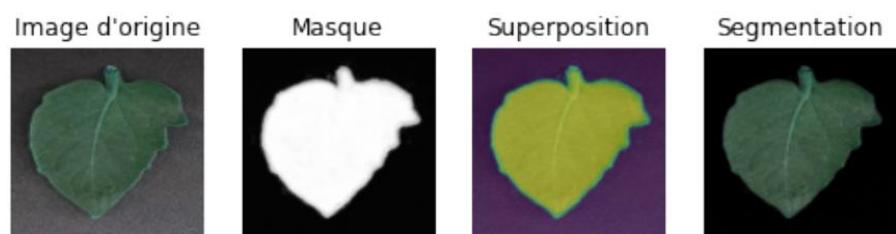
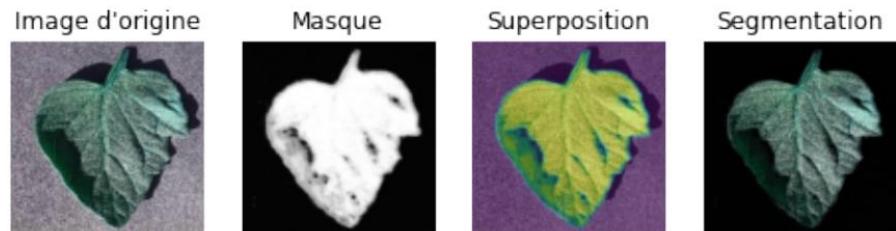


Superposition



Segmentation





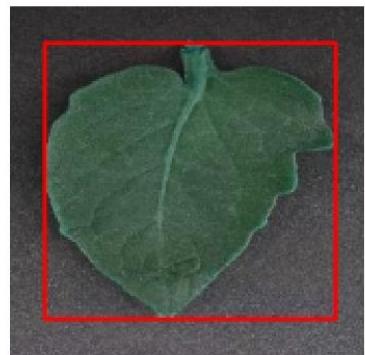
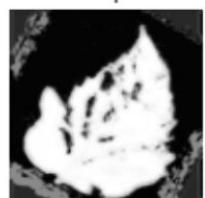


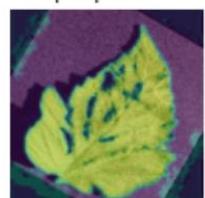
Image d'origine



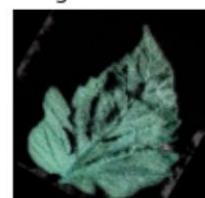
Masque



Superposition



Segmentation



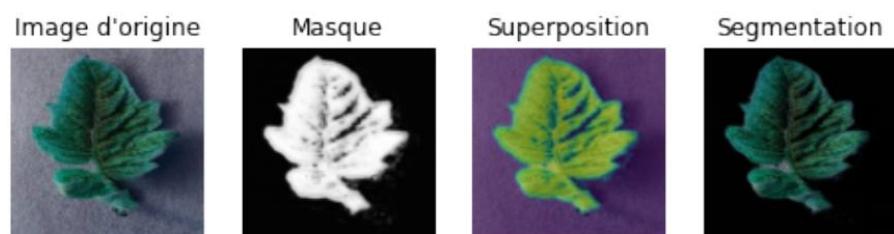
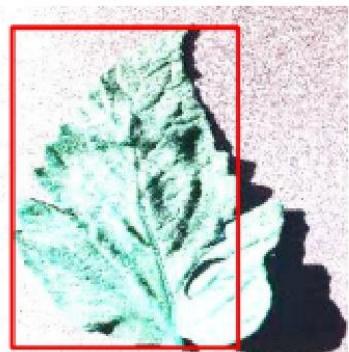
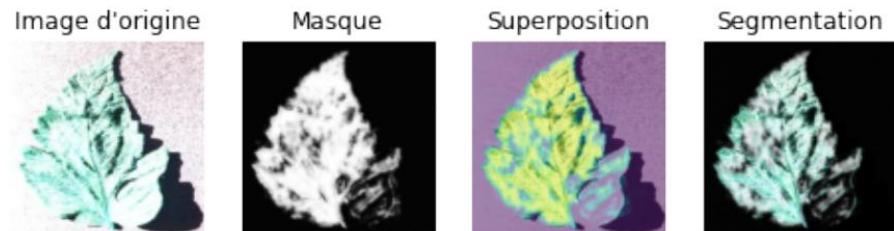
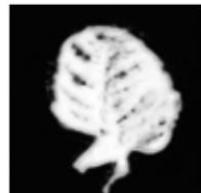




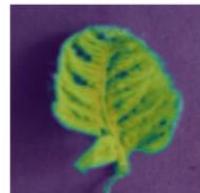
Image d'origine



Masque



Superposition



Segmentation



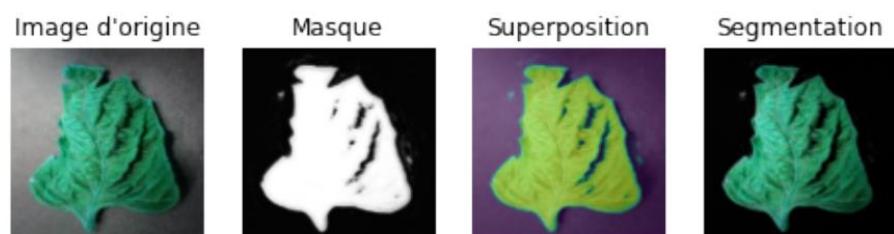
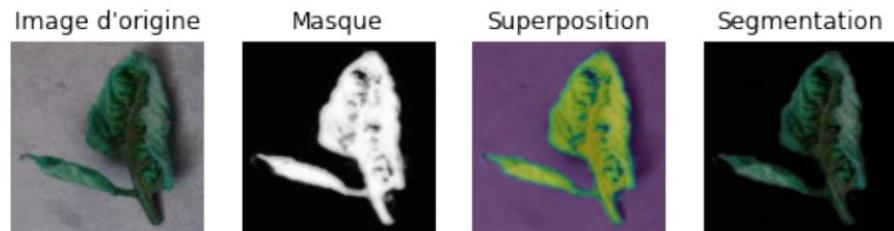




Image d'origine



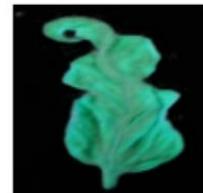
Masque



Superposition



Segmentation



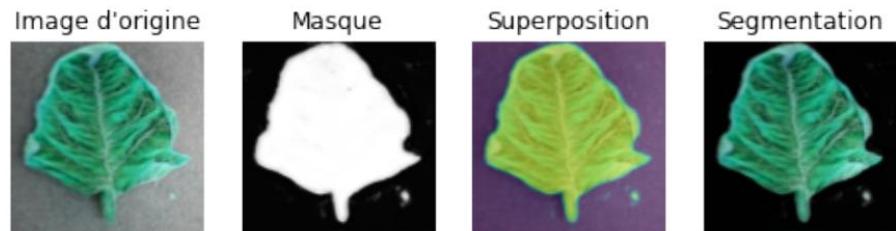


Table des matières

| | |
|-----------------------------------------------------------------|----|
| 1. Contexte | 2 |
| 2. Objectifs..... | 4 |
| 2.1. Fonctionnels | 4 |
| 2.2. Techniques | 4 |
| 2.3. Pédagogiques..... | 4 |
| 2.3.1. Sur le périmètre du projet | 4 |
| 2.3.2. Sur le pilotage du projet..... | 5 |
| 3. Niveau d'expertise sur la problématique adressée | 6 |
| 4. Données | 7 |
| 4.1. Cadre..... | 7 |
| 4.1.1. COCO Common Objects in Context..... | 7 |
| 4.1.2. Open Image Dataset..... | 7 |
| 4.1.3. Kaggle | 7 |
| 4.1.4. Plant Village..... | 7 |
| 4.2. Pertinence | 8 |
| 5. Projet | 11 |
| 5.1. Classification du problème | 11 |
| 5.2. Partie principale du projet | 11 |
| 6. Choix du modèle & Optimisation..... | 13 |
| 6.1. Classification des images..... | 13 |
| 6.1.1. CNN classique et « baseline » | 13 |
| 6.1.2. Architecture basée sur CNN LeNet..... | 14 |
| 6.1.3. Transfer Learning et VGG16..... | 16 |
| 6.1.4. Amélioration des prédictions du CNN | 18 |
| 6.2. Segmentation d'images | 19 |
| 6.2.1. Premières approches..... | 19 |
| 6.2.2. Modèle de Deep Learning..... | 20 |
| 6.2.3. Résultats..... | 22 |
| 6.3. Détection d'objets graphiques ou « Bounding Box » | 22 |
| 7. Description des travaux réalisés | 24 |
| 7.1. Répartition de l'effort sur la durée et dans l'équipe..... | 24 |
| 7.2. Bibliographie | 25 |
| 7.2.1. Scientifique | 25 |
| 7.2.2. Technique..... | 26 |

| | | |
|--------|---------------------------------------------------------------------|-----|
| 7.3. | Difficultés rencontrées lors du projet | 26 |
| 7.3.1. | Timing Projet - Formation | 26 |
| 7.3.2. | Ressources techniques..... | 27 |
| 7.3.3. | Echange d'information dans le code et protocoles fonctionnels | 27 |
| 7.3.4. | Environnements hétérogènes | 28 |
| 8. | Succès du projet et évolutions possibles..... | 29 |
| 8.1. | Objectifs initiaux..... | 29 |
| 8.2. | Facteurs de validation..... | 29 |
| 8.3. | Résultats du projet..... | 29 |
| 8.3.1. | Traiter une photographie soumise par l'utilisateur..... | 29 |
| 8.3.2. | Identifier une plante dans une image | 29 |
| 8.3.3. | Identifier les maladies possibles | 29 |
| 8.3.4. | Retourner une description à l'utilisateur | 30 |
| 8.3.5. | Extraire la plante de l'image..... | 30 |
| 8.3.6. | Valorisation des résultats..... | 30 |
| 8.4. | Perspectives d'évolutions possibles..... | 30 |
| 8.4.1. | Robustesse du modèle | 31 |
| 8.4.2. | Domaine de prédiction | 31 |
| 8.4.3. | Gamme de services | 31 |
| 9. | Bilan & Conclusion..... | 32 |
| 9.1. | Bilan scientifique | 32 |
| 9.2. | Bilan technique et professionnel | 32 |
| 9.3. | Bilan de l'équipe | 32 |
| 9.3.1. | Investissement..... | 32 |
| 9.3.2. | Timing..... | 32 |
| 9.3.3. | Homogénéité du groupe et complémentarité des compétences | 32 |
| 9.4. | Conclusion..... | 33 |
| 10. | Annexes | 34 |
| 10.1. | Dataviz des jeux de données..... | 34 |
| 10.2. | Réduction de dimension sur les images | 51 |
| 10.3. | CNN VGG16 & Tranfer learning | 59 |
| 10.4. | Segmentation..... | 76 |
| 10.5. | Identification d'objet et « bounding box » | 90 |
| | Table des matières | 119 |
| | Table des illustrations | 121 |

Table des illustrations

| | |
|---------------------------------------------------------------------------------------------------------|----|
| <i>Figure 1 - Répartition des photos par classe</i> | 8 |
| <i>Figure 2 - Répartition des photos par couleur</i> | 8 |
| <i>Figure 3 - Image moyenne de chaque classe</i> | 9 |
| <i>Figure 4 - Nombre d'images par classe</i> | 9 |
| <i>Figure 5 - CNN classique servant de baseline</i> | 14 |
| <i>Figure 6 - CNN classique : courbe de performance et overfitting</i> | 14 |
| <i>Figure 7 - Structure CNN type LeNet</i> | 15 |
| <i>Figure 8 - CNN type LeNet - Courbe de performance</i> | 16 |
| <i>Figure 9 - VGG16 - Entrainement sur partie classification</i> | 17 |
| <i>Figure 10 - VGG16 - Entrainement sur partie classification et couches de convolution supérieures</i> | 17 |
| <i>Figure 11 - VGG16 - Courbe de performance sur les dernières époques</i> | 18 |
| <i>Figure 12 - VGG16 - Courbes de perte et de performance</i> | 18 |
| <i>Figure 13 - VGG16 - Exemple de prédictions d'images de plantes</i> | 18 |
| <i>Figure 14 - VGG16 - Exemple de prédictions images de plantes et autres</i> | 19 |
| <i>Figure 15 - Transformation des images segmentées en masques</i> | 20 |
| <i>Figure 16 - Exemple de masque généré à partir d'une image segmentée</i> | 21 |
| <i>Figure 17 - Structure du modèle U-Net</i> | 21 |
| <i>Figure 18 - Exemple de prédictions de masques du modèle, en comparaison avec des labels</i> | 22 |
| <i>Figure 19 - Détection d'objet - Bounding box</i> | 23 |
| <i>Figure 20 - Diagramme de Gantt - Partie 1</i> | 25 |
| <i>Figure 21 - Diagramme de Gantt - Partie 2</i> | 25 |