

# Rapport Projet Final

PIF 6004

Deep Learning

Session Hiver 2021

## Classification des maladies des feuilles de manioc

Identifier le type de maladie présente sur une image de feuille de manioc



# Rapport Projet Final

**PIF 6004**  
**Deep Learning**  
**Session Hiver 2021**

Classification des maladies des feuilles de manioc  
Identifier le type de maladie présente sur une  
image de feuille de manioc

Par

Nom	Prénom
Ba	Seydou
Ndiaye	Mamadou moctar

Institution : UNIVERSITÉ DU QUÉBEC À TROIS-RIVIÈRES  
Professeur Assistant : Francois Meunier/Usef Faghihi  
Département : Mathématiques et Informatique  
Lieu : Ringuet  
Durée Projet : Janvier, 2021 - Avril, 2021

UQTR



**Savoir.**  
**Surprendre.**

# Table des matières

<b>Liste des figures</b>	<b>ii</b>
<b>Liste des tableaux</b>	<b>iii</b>
<b>1 Introduction</b>	<b>1</b>
<b>2 Catégorie des maladies</b>	<b>2</b>
2.1 CGM (Cassava Green Mottle) . . . . .	3
2.2 CBSD(Cassava Brown Streak Disease) . . . . .	4
2.3 CMD(Cassava Mosaic Disease). . . . .	5
2.4 CBB(Cassava Bacterial Blight) . . . . .	6
<b>3 Travaux Connexes</b>	<b>7</b>
<b>4 Méthodes Proposées</b>	<b>8</b>
4.1 Prétraitement des données . . . . .	8
4.1.1 Importation des bibliothèques . . . . .	8
4.1.2 Extraction des données (zip) . . . . .	9
4.2 Séparations et augmentations des données . . . . .	10
4.2.1 Séparation des données . . . . .	10
4.2.2 Augmentations des données . . . . .	11
4.3 Importation de l'ensemble des données (fichier csv). . . . .	12
4.4 Visualisation statistique des données . . . . .	14
4.5 Classification des images . . . . .	16
<b>5 Configurations expérimentales</b>	<b>17</b>
5.1 Architecture modèle . . . . .	17
5.1.1 Modèle ResNet50 . . . . .	17
5.1.2 Modèle EfficientNetB5 . . . . .	18
5.2 Entraînement du modèle . . . . .	19
<b>6 Résultats et Discussions</b>	<b>20</b>
<b>7 Conclusions</b>	<b>26</b>
<b>Références</b>	<b>27</b>

# Liste des figures

2.1	Catégories des maladies courantes telles qu'elles sont visualisées par les yeux humains à partir de l'ensemble de données. . . . .	2
2.2	Infections de la maladie du Marbure vert du manioc . . . . .	3
2.3	infections de la maladie de la striune brune du manioc . . . . .	4
2.4	Infections de la maladie du Marbure vert du manioc . . . . .	5
2.5	infections de la maladie de la striune brune du manioc . . . . .	6
4.1	importation des bibliothèques . . . . .	8
4.2	Infections de la maladie du Marbure vert du manioc . . . . .	9
4.3	Séparation de nos données en test train et validation . . . . .	10
4.4	Séparation de nos données en test train et validation . . . . .	11
4.5	augmentation de nos données dans la dataset . . . . .	11
4.6	augmentation de nos données dans la dataset . . . . .	12
4.7	ensemble des images dans le fichier csv . . . . .	13
4.8	nom des labels . . . . .	13
4.9	nombre d'images pour chaque maladie . . . . .	14
4.10	Histogramme des maladies . . . . .	14
4.11	correspondance entre image et label . . . . .	15
4.12	Classification des images par label . . . . .	16
5.1	architecture du modèle ResNet50 . . . . .	17
5.2	architecture du modèle EfficientNetB5 . . . . .	18
6.1	Report de classification des effets des différentes taux d'apprentissage de SGD sur la précision du modèle Resnet50. . . . .	20
6.2	Report de classification des effets des différentes taux d'apprentissage de ADAM sur la précision du modèle Resnet50. . . . .	21
6.3	Report de classification de l'effet de la classe weight sur la précision du modèle Resnet50. . . . .	21
6.4	Différence sur la précision du modèle Resnet50 sans augmentation et avec augmentation. . . . .	22
6.5	figures montrant la précision sur le modèle EfficientNetB5 . . . . .	23
6.6	Effets des poids sur la précision sur le modèle EfficientNetB5 . . . . .	24
6.7	Différence sur la précision du modèle EfficientNetB5 sans augmentation et avec augmentation. . . . .	24

# Liste des tableaux

5.1	Tableau des Optimiseurs . . . . .	19
6.1	Effet des différentes taux d'apprentissage de SGD sur la précision du modèle Resnet50. .	20
6.2	Effet des différentes taux d'apprentissage de SGD sur la précision du modèle Resnet50. .	21
6.3	Tableau des différentes transformations de nos données et leur précision pour le modèle RestNet50. . . . .	22
6.4	Tableau des différentes transformations de nos données et leur précision pour le modèle EfficientNetB5. . . . .	24

# Introduction

Le manioc (*Manihot esculenta*), plus grand producteur de glucides que les principales cultures céréalières comme le riz et le maïs est une nourriture de base introduite en Afrique au 16<sup>e</sup> siècle. Les tubercules de manioc peuvent être gardés dans le sol jusqu'à deux ans avant d'être récoltés. Les feuilles de manioc et les racines féculentes sont les plus consommées; les racines féculentes sont utilisées comme source d'énergie et peuvent être consommées crues, rôties, bouillies ou traitées de différentes façons pour la consommation humaine. Les feuilles de manioc source de protéines, de vitamines et de minéraux sont beaucoup consommées dans les pays subsahariens. Leur utilisation diffère selon les pays au Nigéria les principaux produits alimentaires pour la consommation humaine sont le gari, le fufu et le lafun [4] [12]. Le manioc apporte un grand rendement dans des meilleures conditions, fonctionne mieux que les autres dans des conditions sous optimales et aussi peut être cultivé dans des sols pauvres mais le principal problème est que le manioc est vulnérable face à certaines maladies et aussi à des souches virales et bactériennes [11]. En Ouganda, Les plus courantes sont le CMD qui présente des symptômes de jaunissement et de rides des feuilles et le CBB qui pourrit les racines font parti des graves menaces de la sécurité alimentaire en Afrique subsaharienne. La CMD a été signalée pour la première fois en Tanzanie vers la fin du 20<sup>e</sup> siècle et à partir de ce moment il s'est propagé dans les autres pays causant de grandes pertes économiques et une famine dévastatrice [7] [8].

Avec l'avènement de nouvelles technologies et aussi l'enjeu de l'intelligence artificielle, beaucoup de méthodes de détections de maladies ont vu leur existence cela justifiant l'appel des agriculteurs avec l'aide des experts pour diagnostiquer et visualiser les plantes et aussi apporter de l'aide aux agriculteurs. Cela est une main-d'œuvre intense pour faire face à ce fléau, l'utilisation de nouvelles technologies permettant aux agriculteurs d'accéder aux applications via leur mobiles pour visualiser et détecter les plantes souffrant de maladies. L'utilisation de notre modèle montre des résultats meilleurs permettant aux agriculteurs de détecter rapidement la propagation des maladies avec l'utilisation de deux modèles à savoir le modèle RestNet50 et le modèle EfficientNetB5.



# 2

## Catégorie des maladies

Les maladies représentent la contrainte majeure de la production africaine de manioc. La plante se trouve affectée par quatre maladies courantes indiquées dans la figure ci-dessous. Ses maladies ont des symptômes uniques et qui apparaissent différemment sur la feuille. Cela peut être utile pour classer et différencier ces maladies selon leur type.



**FIGURE 2.1.** Catégories des maladies courantes telles qu'elles sont visualisées par les yeux humains à partir de l'ensemble de données.

## 2.1. CGM (Cassava Green Mottle)

Plus connu sous le nom de Marbrure vert du Manioc, il est uniquement connu des îles Salomon. Il a été découvert pour la première fois sur Choiseul des années 1970. Plus récemment (2010), des symptômes similaires ont été observés sur Malaita. Les jeunes feuilles sont plissées avec des taches jaunes pales à distinct (fig.1) , des motifs verts (mosaïques) et des marges tordues (fig.2). Habituellement, les pousses récupèrent des symptômes et semblent saines. Parfois, les plantes sont gravement rabougries, les racines comestibles sont absentes ou, si elles sont présentes, elles sont petites et ligneuses lorsqu'elles sont cuites. En laboratoire, le virus peut être transmis entre les plantes dans la sève et aussi dans les graines. Trente pour cent des graines de plants de tabac infectés étaient infectées. On ignore si elle se propage également dans les graines de manioc. La graine n'est pas utilisée pour la culture du manioc, donc la propagation sur choiseul est plus probable dans les boutures malades. Cependant, il existe d'autres possibilités. Dans un essai sur Choiseul, 4% des plants issus de boutures prélevées sur Guadalcanal sont tombés malades dans les 250 jours suivant la plantation. La façon dont cette infection s'est produite est inconnue, mais il existe plusieurs possibilités. Les plantes qui se touchent dans le vent peuvent avoir transféré la sève entre celles infectées et saines. Il existe également une possibilité de propagation par les nématodes et le pollen. On pense que le virus appartient au groupe des népovirus et que les membres de ce groupe se propagent de cette manière.



(a) Figure 1



(b) Figure 2

**FIGURE 2.2.** Infections de la maladie du Marbure vert du manioc



## 2.2. CBSD(Cassava Brown Streak Disease)

La striure brune du manioc (CBSD) affecte tous les organes de la plante qui développent différents types des symptômes dont les syndromes qui sont manifestés ainsi que leur degré de développement varient considérablement suivant les conditions environnementales et le stade de croissance de la culture en relation avec la période d'infection et la sensibilité du cultivateur. La mosaïque et la striure brune du manioc entraînent toutes les deux la marbrure chlorotique des feuilles bien que les symptômes dus à ces maladies sont très distincts lorsque chacune se développe séparément. Dans certaines conditions environnementales non encore identifiées les symptômes foliaires de la striure brune du manioc peuvent être inexistantes sur les plantes qui sont atteintes et ceci se remarque surtout sur les feuilles qui sont nouvellement formées après une défoliation due à la sécheresse.

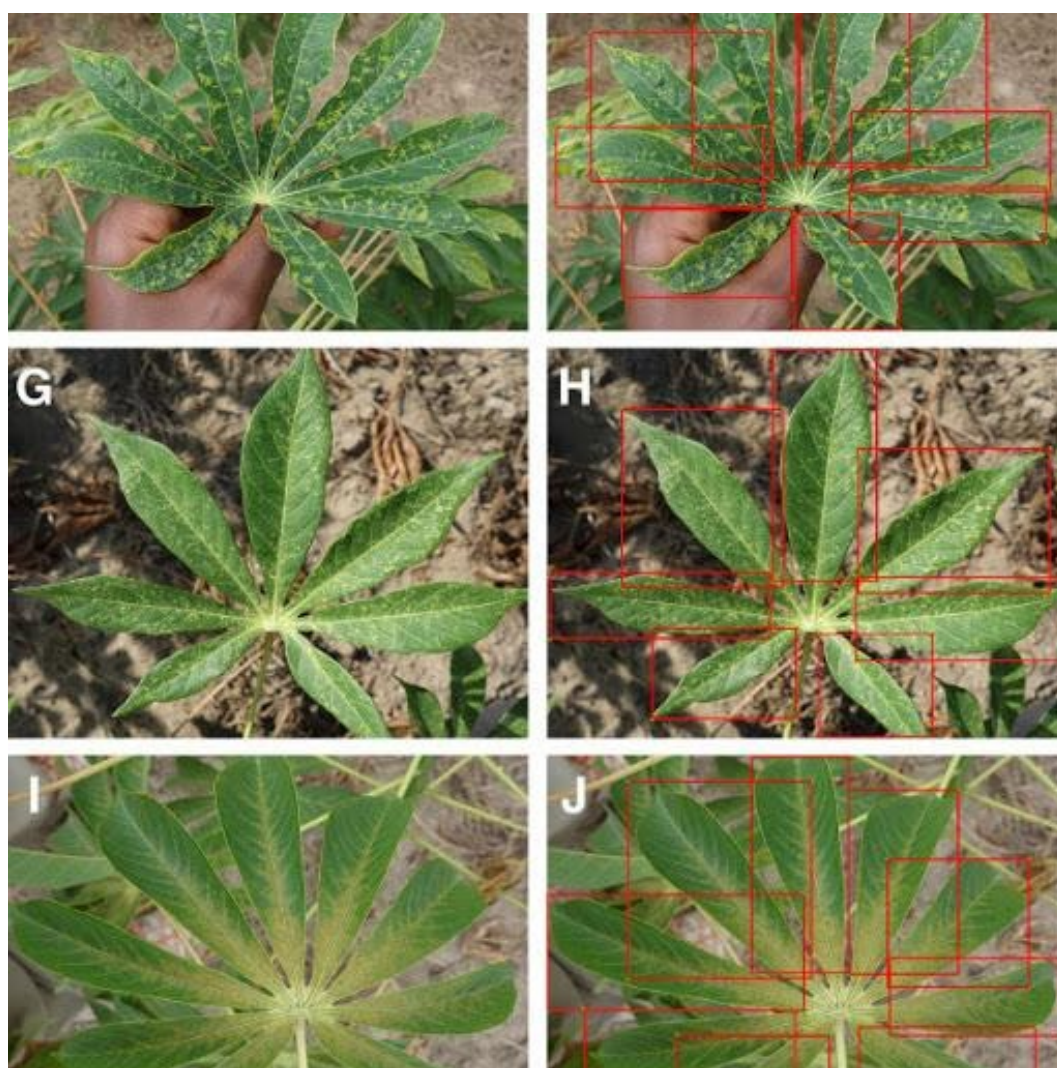
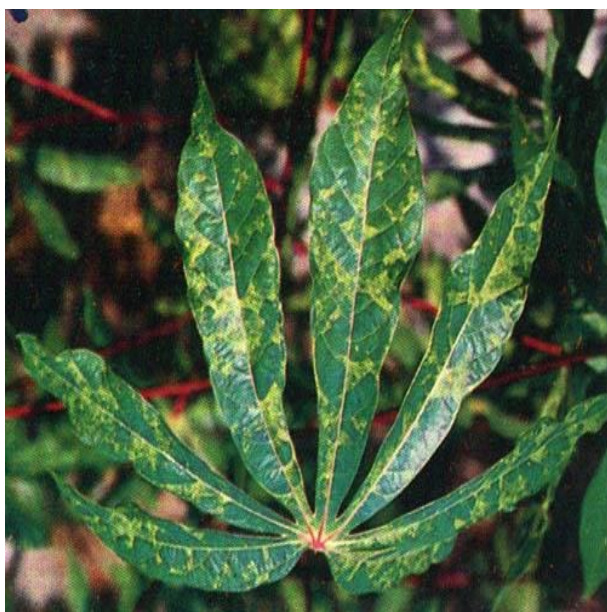


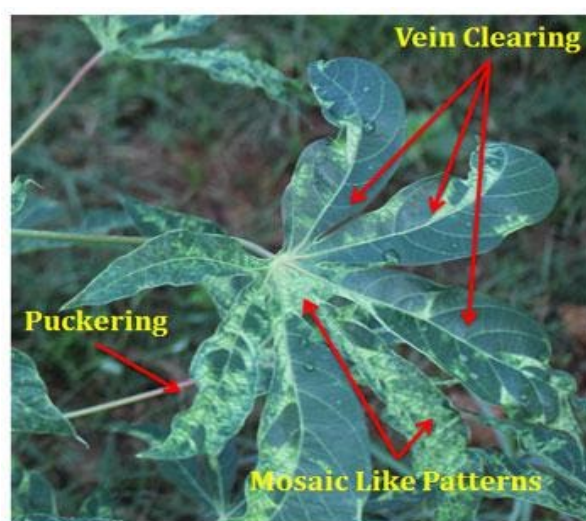
FIGURE 2.3. infections de la maladie de la striure brune du manioc

## 2.3. CMD(Cassava Mosaic Disease)

Communément appelé la mosaïque africaine du manioc, cette maladie était décrite pour la première fois en 1894 là où c'est actuellement connue comme Tanzanie. Les symptômes de la maladie apparaissent comme une mosaïque caractéristique de feuilles affectant des parties bien discrètes. Ils peuvent être discernés au stade initial de développement de la feuille (Fig. 1a). Les parties chlorotiques n'étendent pas entièrement de manière que les contraintes causées par l'expansion inégale des parties laminaires entraînent une malformation et une distorsion du limbe (Fig. 1b). Les feuilles sévèrement atteintes sont réduites, déformées et recroquevillées, et elles sont caractérisées par la présence des portions jaunâtres alternées des parties vertes normales. Les plantes affectées sont en général La chlorose des feuilles apparait d'une coloration pâle-jaunâtre à blanchâtre. Les parties affectées sont clairement démarquées et varient en dimension allant de toute la feuille entière aux petites taches. La foliole peut aussi développer une mosaïque uniforme ou localisée à quelques endroits souvent à la base. La distorsion et la réduction en dimension des feuilles ainsi que la croissance générale retardée des plantes apparaissent comme étant des effets secondaires associés à la sévérité des symptômes. Souvent ces symptômes foliaires varient selon la feuille, la tige et la plante, ainsi que pour une même variété et souche du virus dans une seule localité. Cette variation est souvent attribuée aux différentes souches de virus, à la susceptibilité de l'hôte, à l'âge de la plante et aux facteurs de l'environnement comme la fertilité et l'humidité du sol, la radiation solaire et la température. Normalement les températures froides favorisent l'expression des symptômes tandis que les températures chaudes les atténuent.



(a) Figure 1



*Symptoms of Cassava Mosaic Disease on Leaves*

(b) Figure 2

**FIGURE 2.4.** Infections de la maladie du Marbure vert du manioc



## 2.4. CBB(Cassava Bacterial Blight)

La brûlure bactérienne du manioc a été décrite pour la première fois en 1912 au Brésil(Bondar,1912).Au cours des années 1970, il a été trouvé dans la plupart des zones de culture du manioc d'Amérique centrale et du Sud, des Caraïbes, d'Afrique et d'Asie ( Bradbury, 1986 ). Les feuilles présentent des taches angulaires (de 1 à 4 mm de diamètre) de couleur vert foncé à bleu, imbibées d'eau, limitées par des veinules et irrégulièrement réparties sur le limbe. Avec le temps, ils s'étendent et fusionnent fréquemment le long des nervures ou des bords de la feuille ; la partie centrale devient brune et la partie imbibée d'eau est souvent entourée d'un halo chlorotique. Les lésions apparaissent comme des taches translucides lorsqu'elles sont vues à contre-jour. Sous une loupe, de petites gouttelettes d'exsudat suintant de la partie centrale de la lésion sont visibles sur la face inférieure des feuilles. Les gouttelettes, qui brillent d'abord blanc crème et ensuite jaunes, sont facilement dissoutes par la pluie ou la rosée ; en séchant, ils forment une fine échelle. Dans des conditions favorables (jeunes feuilles, humidité élevée du sol et de l'air), des tâches ponctuelles imbibées d'eau se développent, dispersées autour de jeunes taches angulaires. La partie environnante du limbe vire au brun clair et en 2-3 jours, de vastes zones des folioles se flétrissent non seulement vers la pointe ou le bord de la foliole, mais également vers la base. Les parties affectées présentent des zones marron clair et vert comme si elles avaient été brûlées superficiellement. Ces zones nécrotiques ne sont pas translucides, aucun exsudat bactérien n'est observé et les bactéries sont absentes ou présentes en quantités très limitées aux bords des lésions de brûlure en extension. Une attaque sévère entraîne un séchage prématuré et une perte des feuilles. Ces zones nécrotiques ne sont pas translucides, aucun exsudat bactérien n'est observé et les bactéries sont absentes ou présentes en quantités très limitées aux bords des lésions de brûlure en extension. Une attaque sévère entraîne un séchage prématuré et une perte des feuilles.



**FIGURE 2.5.** infections de la maladie de la striure brune du manioc

# 3

## Travaux Connexes

Les modèles d'apprentissage profond ont permis de détecter automatiquement certaines maladies chez les plantes en prenant comme donnée d'entrée l'image d'une feuille et de prédire si la feuille est affectée ou pas. Dans ce sillage, Les travaux suivantes [9], [6], [10], [5] ont été réalisées pour diagnostiquer les maladies des plantes en prenant les images de feuilles de plantes comme source de données.

Dans [9], les premiers travaux considère que l'utilisation des feuilles comme donnée clé d'entrée montre l'apparition de symptômes mais en un stade plus avancé. Dans ce cas, pour faire face à cette situation et sauver les plantes affectées, l'étude de la spectroscopie a été un pas vers la solution. La spectroscopie a pour objectif d'étudier comment les matériaux interagissent avec la lumière avec des longueurs d'ondes absorbées ou réfléchies par le matériel une fois exposé aux rayons lumineux. Son objectif est d'étudier comment les maladies de la plante se comporte avec l'interaction de la lumière tout en détectant et arrêtant la propagation des infections à un stade prématuré.

[6], l'approche utilisée permet de se concentrer sur l'identification et la détection de maladies et des ravageurs sur les plantes de tomates tout en classer les maladies et la reconnaissance de la zone affectée. Cette approche utilise les réseaux de neurones de l'apprentissage profond pour pouvoir détecter et reconnaître les zones affectées en tant réel mais aussi de détecter la maladie à un stade prématuré. Son objectif est de se passer des autres méthodes traditionnelles telles les méthodes de collectes physiques et aussi des autres méthodes telles la spectroscopie et l'imagerie comme évoquées dans les travaux antérieurs mais de se concentrer sur la méthode chimique tout en analysant ses plantes en laboratoire en créant des applications robustes basées sur l'apprentissage profond.

Dans [10], l'approche utilisée est l'apprentissage par transfert en utilisant le modèle Inception v3 implémenté dans Tensorflow pour détecter et classer les maladies de manioc en trois types et en deux dommages causés par les parasites dans un ensemble d'images prises en Tanzanie. Ce modèle a donné de meilleurs résultats pour la détection automatique des maladies et aussi elle a été déployée dans des appareils mobiles pour faciliter la détection des maladies et aussi des parasites sur les feuilles de manioc.

Dans [5], l'auteur Konstantinos P. Ferentinos a utilisé les réseaux de neurones convolutionnelles pour détecter et diagnostiquer les plantes affectées en utilisant une base de données de 87848 images contenant 25 plantes différentes avec un ensemble de 58 classes différentes combinant les maladies de plantes et aussi des plantes saines. Plusieurs modèles de réseaux convolutionnelles ont été formés telles que AlexNet, AlexNetOWTBn, GoogLeNet, Overfeat et VGG avec une meilleure précision de 99,53% faisant du modèle un meilleur pour détecter automatiquement les maladies.

Notre étude se base sur les travaux évoqués en utilisant les architectures des modèles des réseaux de neurones convolutionnelles telles que ResNet50 et EfficientNetB5.

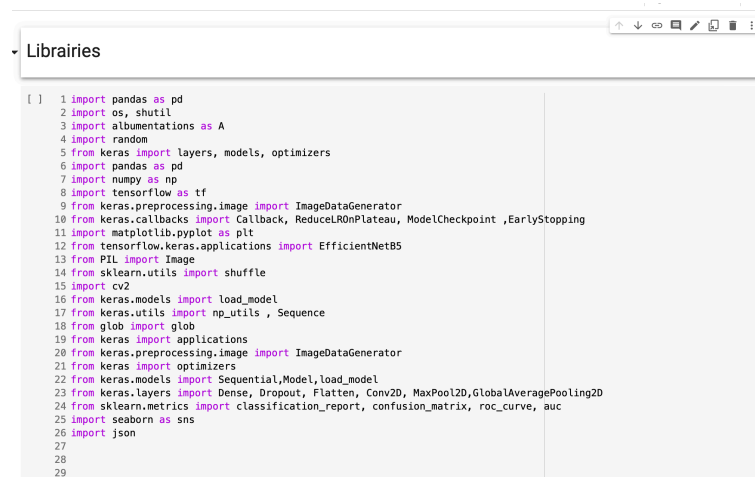
# Méthodes Proposées

## 4.1. Prétraitement des données

Le prétraitement des données est une technique d'exploration de données qui consiste à transformer des données brutes dans un format compréhensible. Les données réelles sont souvent incomplètes, incohérentes et / ou absentes de certains comportements ou tendances, et sont susceptibles de contenir de nombreuses erreurs. Le prétraitement des données est une méthode éprouvée pour résoudre ces problèmes.[1]

### 4.1.1. Importation des bibliothèques

L'industrie informatique accélère le développement de machines intelligentes, capables de présenter un comportement humain en matière d'apprentissage. Cette simulation de l'intelligence humaine s'appuie sur diverses bibliothèques Python spécialement conçues pour dynamiser et optimiser cette branche de l'informatique.



```
[ ] 1 import pandas as pd
2 import os, shutil
3 import albumentations as A
4 import random
5 from keras import layers, models, optimizers
6 import pandas as pd
7 import numpy as np
8 import tensorflow as tf
9 from keras.preprocessing.image import ImageDataGenerator
10 from keras.callbacks import Callback, ReduceLROnPlateau, ModelCheckpoint, EarlyStopping
11 import matplotlib.pyplot as plt
12 from tensorflow.keras.applications import EfficientNetB5
13 from PIL import Image
14 from sklearn.utils import shuffle
15 import cv2
16 from keras.models import load_model
17 from keras.utils import np_utils, Sequence
18 from glob import glob
19 from keras import applications
20 from keras.preprocessing.image import ImageDataGenerator
21 from keras import optimizers
22 from keras.models import Sequential, Model, load_model
23 from keras.layers import Dense, Dropout, Flatten, Conv2D, MaxPool2D, GlobalAveragePooling2D
24 from sklearn.metrics import classification_report, confusion_matrix, roc_curve, auc
25 import seaborn as sns
26 import json
27
28
29
```

FIGURE 4.1. importation des bibliothèques

En premier lieu nous avons importé les bibliothèques suivantes :

- matplotlib.pyplot : pour créer des visualisations statistiques et interactive
- numpy : pour la manipulation de matrices et de tableaux multidimensionnels
- tensorflow : qui est une bibliothèque de machine learning pour développer des -algorithmes d'apprentissages machine



- glob : cherche tous les chemins correspondants à un motif particulier
- pandas : pour la manipulation et l'analyse des données
- keras : framework de deep-learning, pratique pour réduire les charges cognitives.
- Albumentation : est une bibliothèque d'augmentations d'images qui nous permettra de modifier la position des images, le contraste, la largeur etc.

#### 4.1.2. Extraction des données (zip)

Pour ce projet nous travaillons sur Google colab qui est un service Cloud basé sur Jupiter notebook et qui nous permet d'entraîner des modèles directement dans le Cloud. La compression de fichiers permet de réduire leur taille. Les données exportées sont au format zip ce qui nous permettra de télécharger plus rapidement nos données. Après téléchargement la prochaine phase consiste à décompresser les données pour pouvoir les manipuler.

```

1 #Download dataset (zip file)
2 # Gdrive link of the dataset
3 #link = 'https://drive.google.com/file/d/1bi2-gPhYnUAYn2AH7jyHxkWC6YGz0Wur/view?usp=share'
4 #link = 'https://drive.google.com/file/d/1u-DU2gB3Bv6uoZBhEFN43kUU_d0RKX2e/view?usp=share'
5 #link = 'https://drive.google.com/file/d/1Tiu8Z1iI4iwGDDXtZHGfcmKjKv_BthNO/view?usp=share'
6 #_, id = link.split('=')
7 #id = "1bi2-gPhYnUAYn2AH7jyHxkWC6YGz0Wur"
8 #id = "1u-DU2gB3Bv6uoZBhEFN43kUU_d0RKX2e"
9
10 id_data = "1u-DU2gB3Bv6uoZBhEFN43kUU_d0RKX2e"
11 id_train_images= "1Tiu8Z1iI4iwGDDXtZHGfcmKjKv_BthNO"
12 #path_data_test = 'data_test.zip'
13 path_data = 'data.zip'
14 path_train_images = 'train_images.zip'
15
16 # Download if data.zip file does not exist
17 if os.path.isfile(path_data):
18     print("archive data already downloaded")
19 else:
20     print("Downloading data dataset (data.zip)...")
21     downloaded = drive.CreateFile({'id':id_data})
22     downloaded.GetContentFile(path_data)
23     print("Download data finished")
24
25 # Download if train_images.zip file does not exist
26 if os.path.isfile(path_train_images):
27     print("archive train_images already downloaded")
28 else:
29     print("Downloading train_images dataset (train_images.zip)...")
30     downloaded = drive.CreateFile({'id':id_train_images})
31     downloaded.GetContentFile(path_train_images)
32     print("Download train_images finished")
33
34 !ls

```

(a) Figure 1

```

[ ] 1 #Extract dataset archive
2
3 # Extract dataset archive
4 import zipfile
5
6 EXTRACT_PATH = "."
7
8 #extract data.zip
9 archive = zipfile.ZipFile(path_data, 'r')
10 archive.extractall(path=EXTRACT_PATH)
11 if os.path.isdir(EXTRACT_PATH):
12     print("archive data successfully extracted")
13 else:
14     print("Can't extract archive data")
15
16 #extract train_images.zip
17 archive = zipfile.ZipFile(path_train_images, 'r')
18 archive.extractall(path=EXTRACT_PATH)
19 if os.path.isdir(EXTRACT_PATH):
20     print("archive train_images successfully extracted")
21 else:
22     print("Can't extract archive train_images")
23
archive data successfully extracted
archive train_images successfully extracted

```

(b) Figure 2

FIGURE 4.2. Infections de la maladie du Marbure vert du manioc

## 4.2. Séparations et augmentations des données

L'entraînement d'un réseau de neurones profond sur très peu d'images est souvent challengeant : le modèle n'ayant accès qu'à un nombre limité d'observations, il va avoir tendance à faire de "l'overfitting", c'est à dire sur-apprendre à partir de l'échantillon d'entraînement, sans pour autant être capable d'émettre des prédictions pertinentes sur de nouvelles images – dans ce cas les performances sont faibles sur l'échantillon de test alors qu'elles étaient bonnes sur les images d'entraînement. Ce phénomène est bien connu des Data Scientists, qui le résolvent souvent en augmentant la taille du dataset et/ou réduisant le nombre de paramètres du modèle.

### 4.2.1. Séparation des données

C'est une technique qui transforme les données brutes en un format compréhensible. Les données du monde réel (données brutes) sont toujours incomplètes et ces données ne peuvent pas être envoyées via des modèles car cela entraînerait certaines erreurs. C'est pourquoi nous devons prétraiter les données avant de les envoyer via un modèle[2]. On redimensionne les images en 128x128, le nombre de classes (5 classes) et le nombre de batch(nombre d'époque) qui supérieur au nombre d'images dans notre dataset pour permettre à l'algorithme d'utiliser une seule fois l'ensemble des données. Et on fait appel aux fonctions train-datagen et test-datagen qui prend en paramètre le target-size (pour la taille de l'image), color-mode (pour le type de l'image couleur ou gris), shuffle( pour obtenir des images différents à chaque exécution) et d'autres paramètres pour le pré-traitement afin de rendre les données plus compréhensibles comme indiqué à la figure 4.3.

```

1 #ensemble de donnees train et test
2 img_width = 128
3 img_height = 128
4 num_classes = 5
5 batch_size = 22000
6 train_generator = train_datagen.flow_from_directory(directory = train_dir,
7                                                    target_size=(img_width,img_height),
8                                                    color_mode="rgb",
9                                                    batch_size= batch_size,
10                                                   class_mode='categorical',
11                                                   subset='training',
12                                                   shuffle = True,
13                                                   seed=42 ,
14                                                   )
15
16 valid_generator = train_datagen.flow_from_directory(directory = train_dir,
17                                                    target_size=(img_width,img_height),
18                                                    color_mode="rgb",
19                                                    batch_size= batch_size,
20                                                    class_mode='categorical',
21                                                    subset='validation',
22                                                    shuffle = True,
23                                                    seed=42)
24
25 test_generator = test_datagen.flow_from_directory(directory = test_dir,
26                                                    target_size=(img_width,img_height),
27                                                    color_mode="rgb",
28                                                    batch_size = batch_size,
29                                                    class_mode='categorical',
30                                                    shuffle = False,
31                                                    seed=42,
32                                                    )

```

Found 13698 images belonging to 5 classes.  
Found 3422 images belonging to 5 classes.  
Found 4277 images belonging to 5 classes.

FIGURE 4.3. Séparation de nos données en test train et validation

Pour visualiser les séparations effectuées on fait appel à la méthode (.shape) comme indiqué à la figure 4.4 et les résultats sont les suivants. 13698 pour les images d'entraitements. 3422 pour les images de validations 4277 pour les images de test

```

1 #charger tous mes donnees sur ma ram pour avoir un fit avec un temps rapide avec la fonction next()
2 #donnees sans aug utilisable lors du fit
3 x_train, y_train = train_generator.next()
4 x_val, y_val = valid_generator.next()
5 x_test, y_test = test_generator.next()
6 print("Train", x_train.shape)
7 print("Validation", x_val.shape)
8 print("Test", x_test.shape)
9 print("y_train", y_train.shape)
10 print("y_validation", y_val.shape)
11 print("y_test", y_test.shape)
12
Train (13698, 128, 128, 3)
Validation (3422, 128, 128, 3)
Test (4277, 128, 128, 3)
y_train (13698, 5)
y_validation (3422, 5)
y_test (4277, 5)

```

FIGURE 4.4. Séparation de nos données en test train et validation

### 4.2.2. Augmentations des données

On procède à une augmentation des données. L'entraînement d'un réseau de neurones profond sur très peu d'images est souvent challengeant : le modèle n'ayant accès qu'à un nombre limité d'observations, il va avoir tendance à faire de "l'overfitting", c'est à dire le sur-apprentissage[3]. Le modèle aura tendance à apprendre d'autres éléments dans les images qu'il reçoit par.ex quand il apprend sur des images de chiens il arrivera un moment ou il va essayer d'apprendre d'autres éléments de l'image et finira par se perdre. C'est pour cela que notre modèle ne sera pas capable d'émettre des prédictions pertinentes sur de nouvelles images. Dans ce cas les performances sont faibles sur l'échantillon de test alors qu'elles étaient bonnes sur les images d'entraînement. Pour éviter ce problème on augmente nos images en proposant à notre réseau de neurones d'autres images par exemple des images inclinées des images horizontales des images avec de faibles luminosités. Et sans avoir besoin de modifier les images, Keras nous propose une bibliothèque ImageDataGenerator pour effectuer automatiquement les modifications. Comme décrit à la partie importation des bibliothèques, la librairie Albumentations (import albumentations as A) on fait appel à cette fonction pour spécifier les paramètres d'augmentations ( la dimension, la rotation, le contraste...) voir figure 4.5.

```

1 #creation de cette classe pour envoyer des batch contenant des images augmentees
2 class CustomSequence(tf.keras.utils.Sequence):
3
4     def __init__(self, data_x, data_y, batch_size, shuffle=False,trans=False):
5         """Custom data generator for model.fit_generator()
6
7         Params:
8             data_x - np.ndarray with features, shape (dataset_size, ...)
9             data_y - np.ndarray with targets, shape (dataset_size, ...)
10            batch_size - mini-batch size
11            shuffle - shuffle features/targets between epochs
12
13        Note:
14            this class 'shuffle' param will shuffle all examples between epochs
15            model.fit_generator(shuffle=...) param shuffles order of mini-batches,
16            but does not touch what is inside mini-batch
17
18        """
19        self.data_x = data_x
20        self.data_y = data_y
21        self.batch_size = batch_size
22        self.indices = np.arange(len(self.data_x))
23        self.shuffle = shuffle
24        self.trans = trans
25
26        self.transform = A.Compose([
27            A.HorizontalFlip(p=0.5),
28            A.VerticalFlip(p=0.5),
29            A.ShiftScaleRotate(shift_limit=0.0625, scale_limit=0.50, rotate_limit=45, p=.75),
30            A.RandomBrightnessContrast(brightness_limit=0.2, contrast_limit=0.2, p=0.5),
31            A.RandomCrop(width=128, height=128, always_apply=False, p=1.0),
32        ])
33

```

FIGURE 4.5. augmentation de nos données dans la dataset

Après avoir défini ces paramètres on crée la fonction transformed (figure 4.6) pour la l'augmentation d'une image. Dans cette fonction on fera appel à la méthode transform(figure 4.5) qui contient les paramètres de modifications. Ensuite la fonction transform-images(figure 4.6) sera créée pour augmenter les images epoch par epoch. Nous avons adopté cette méthode pour éviter les pertes et les bugs vue la quantité des images très élevées.

```

32
33     self.on_epoch_end()
34
35 def __len__(self):
36     return int(np.ceil(len(self.data_x) / self.batch_size))
37
38 def __getitem__(self, idx):
39     batch_i = self.indices[idx * self.batch_size : (idx + 1) * self.batch_size]
40     batch_x = self.data_x[batch_i]
41     batch_y = self.data_y[batch_i]
42     #print("pre: ", batch_x.shape)
43     if self.trans == True:
44         batch_x = self.transform_images(batch_x)
45     #print("post: ", batch_x.shape)
46     return batch_x, batch_y
47
48
49 def on_epoch_end(self):
50     if self.shuffle:
51
52         np.random.shuffle(self.indices)
53
54
55 def transformed(self, image):
56     augmented_image = self.transform(image=image)['image']
57
58     return augmented_image
59
60 def transform_images(self, batch_images):
61     aug = np.zeros(batch_images.shape)
62     for i in range(batch_images.shape[0]):
63         aug[i] = self.transformed(batch_images[i])
64     return aug
65

```

FIGURE 4.6. augmentation de nos données dans la dataset


### 4.3. Importation de l'ensemble des données (fichier csv)

La plupart des ensembles de données sont au format .csv (valeurs séparées par des virgules). Il est important de conserver l'ensemble de données dans le même dossier que votre programme et de le lire à l'aide d'une méthode appelée read-csv qui se trouve dans la bibliothèque appelée pandas. Comme indiqué à la figure 3.1 notre fichier csv comporte l'ensemble des images ainsi que leur label de correspondance (compris entre 0 et 5). Chaque numéro correspond à un nom de maladie qui sont indiqués dans le fichier label-num-to-disease-map.json voir figure 3.2 On fait appel à la méthode read-csv ensuite spécifier l'emplacement de notre fichier csv afin pouvoir lire le fichier csv.

```

1 DL_dir1 = "/content/drive/My Drive/Projet_PIF6004/"
2
3
4 train_dataset = pd.read_csv(DL_dir1 + "train.csv")
5 train_dataset

```



	image_id	label
0	1000015157.jpg	0
1	1000201771.jpg	3
2	100042118.jpg	1
3	1000723321.jpg	1
4	1000812911.jpg	3
...	...	...
21392	999068805.jpg	3
21393	999329392.jpg	3
21394	999474432.jpg	1
21395	999616605.jpg	4
21396	999998473.jpg	4

21397 rows x 2 columns

FIGURE 4.7. ensemble des images dans le fichier csv

On lit notre fichier JSON(JavaScript Objet Notation) qui contient l'ensemble des noms de chaque label ainsi que leurs étiquettes. figure 4.8

```

[ ] 1 print("label | Name")
2 with open(DL_dir1 + "label_num_to_disease_map.json") as f:
3     label_diseases_name = json.load(f)
4
5 label_diseases_name = {int(k): v for k, v in label_diseases_name.items()}
6 label_diseases_name

```

```

label | Name
{0: 'Cassava Bacterial Blight (CBB)',
1: 'Cassava Brown Streak Disease (CBSD)',
2: 'Cassava Green Mottle (CGM)',
3: 'Cassava Mosaic Disease (CMD)',
4: 'Healthy'}

```

FIGURE 4.8. nom des labels



## 4.4. Visualisation statistique des données

On effectue une visualisation de nos données pour savoir combien d'image nous disposons pour chaque type de maladies. Comme indiqué à la (fig 4.9) on fait appel à la méthode `value_counts()` qui se trouve dans la bibliothèque `pandas` afin d'avoir le nombre total pour chaque type de maladie dans notre dataset. Ensuite avec la fonction `plt.figure()` nous créons un objet figure qui prend comme abscisse le nombre d'image pour chaque maladie du label (0-5) et en ordonnée le des différentes maladies (fig 4.10). La fonction `set-theme` nous permet de définir le thème de notre figure. Et pour afficher notre figure on fait un `plt.show()`.

```
[ ] 1 train_dataset_stat = train_dataset['disease_type'].value_counts()
    2
    3 train_dataset_stat
```

Cassava Mosaic Disease (CMD)	13158
Healthy	2577
Cassava Green Mottle (CGM)	2386
Cassava Brown Streak Disease (CBSD)	2189
Cassava Bacterial Blight (CBB)	1087

Name: disease\_type, dtype: int64

FIGURE 4.9. nombre d'images pour chaque maladie

```
[ ] 1 sns.set_theme(style="darkgrid")
    2 #plt.figure(figsize = (10, 6))
    3 sns.countplot(y=train_dataset['label'].map(label_diseases_name), orient='v')
    4
    5 plt.show()
```

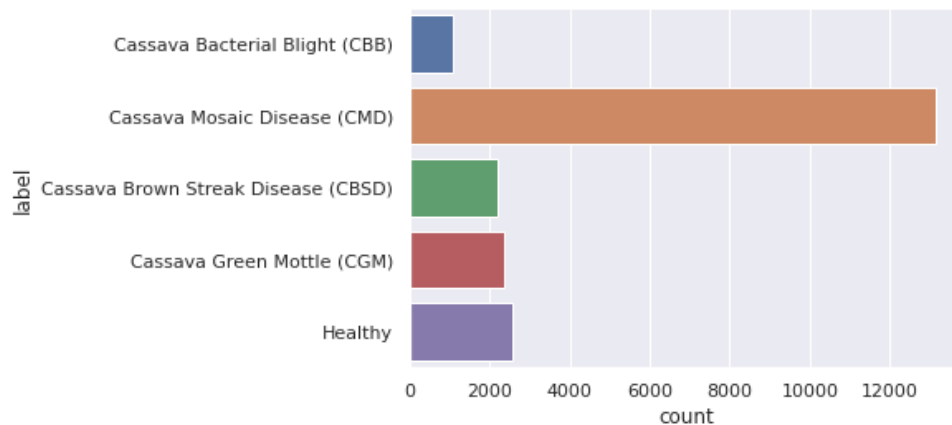


FIGURE 4.10. Histogramme des maladies

Après avoir visualisé nos données, on effectue un mappage (liaison) entre la colonne ['disease-type'] qui comporte le nom des maladies et la colonne ['label'] dans le fichier csv pour en faire qu'un seul tableau comportant à la fois le label et le nom de chaque label (voir figure 4.10) grâce à la fonction `map(label-disease-name)` qui a pris comme argument la variable qui lit le fichier des nom de label.

```
[ ] 1 train_dataset["disease_type"] = train_dataset["label"].map(label_diseases_name)
    2 train_dataset
```

	image_id	label	disease_type
0	1000015157.jpg	0	Cassava Bacterial Blight (CBB)
1	1000201771.jpg	3	Cassava Mosaic Disease (CMD)
2	100042118.jpg	1	Cassava Brown Streak Disease (CBSD)
3	1000723321.jpg	1	Cassava Brown Streak Disease (CBSD)
4	1000812911.jpg	3	Cassava Mosaic Disease (CMD)
...	...	...	...
21392	999068805.jpg	3	Cassava Mosaic Disease (CMD)
21393	999329392.jpg	3	Cassava Mosaic Disease (CMD)
21394	999474432.jpg	1	Cassava Brown Streak Disease (CBSD)
21395	999616605.jpg	4	Healthy
21396	999998473.jpg	4	Healthy

21397 rows x 3 columns

**FIGURE 4.11.** correspondance entre image et label

Comme indiqué à l'image si dessus le fichier comprend désormais le nom de chaque label associé à son étiquette ce qui nous facilitera la suite du travail.

Dans les prochaines sections à venir nous allons classer et afficher les images de notre dataset en fonction de leur catégorie de maladies. Cette approche facilitera la reconnaissance des maladies vue que les images reçues sont dans le désordre.

## 4.5. Classification des images

Les images reçues dans notre dataset n'ont pas été classifiées. Ainsi il est nécessaire de classer les images selon leur type de maladie afin de reconnaître les images. On classe les images en se basant sur le fichier csv qui contient les images ainsi que leurs labels. On définit par la suite le nombre de lignes et de colonnes à afficher (fig.5), la taille des images à afficher ainsi que le chemin vers le fichier des images. Après avoir défini ces on spécifie le label à afficher (ex : 1, 2, 3 ...), le nombre d'image à afficher. Lors de l'affiche du titre de chaque label il est important d'associer chaque étiquette à son nom correspondant. En premier lieu les images sont classées selon leur label par exemple pour le CMD(Cassava Mosaic Disease) l'algorithme classe tout les images donc le label correspond au chiffre trois(3). On suit le même procédé pour classer les autres images avec la catégorie de la maladie comme titre se qui nous permettra de mieux connaître l'appartenance de chaque image.

```


Entrée [11]: 1 def get_image(image_id, labels):
2
3
4     plt.figure(figsize=(20, 18)) # dimensions de l'images à afficher
5
6     for i, (image_id, disease_type) in enumerate(zip(image_id, labels)):
7         plt.subplot(3, 3, i + 1) # nombre de ligne et de colonne des figures et sa position
8         image = cv2.imread(os.path.join(path_input, 'train_images', image_id)) # charger l'image
9         image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB) # couleur de l'image
10
11         plt.imshow(image)
12         plt.title(f'{disease_type}', fontweight='bold', fontsize=12) # titre de chaque image
13         plt.axis("off") # affiche les lignes dans l'image
14
15     plt.show()

Entrée [12]: 1 print("")
2 print("-----Cassava Mosaic Disease (CMD)-----")
3
4 cmd_sample = train_dataset[train_dataset['label'] == 3].sample(6) # nombre d'image à afficher
5 image_ids = cmd_sample['image_id'].values # renvoi les valeurs disponibles dans le dictionnaire donnée
6 labels = cmd_sample['disease_type'].values
7
8 get_image(image_ids, labels) # appel de la fonction get_image

```

-----Cassava Mosaic Disease (CMD)-----

Cassava Mosaic Disease (CMD)



Cassava Mosaic Disease (CMD)



Cassava Mosaic Disease (CMD)




FIGURE 4.12. Classification des images par label

## Configurations expérimentales

### 5.1. Architecture modèle

Un modèle pré-formé est un modèle qui a été formé sur un grand ensemble de données pour résoudre un grand nombre de problèmes de classification similaire à celui que nous voulons résoudre. Ainsi, nous avons utilisé d'abord le modèle ResNet50, ensuite le modèle EfficientNetB5 et enfin on a mis ses deux modèles ensemble pour obtenir plus de performance.

#### 5.1.1. Modèle ResNet50

L'architecture du modèle ResNet50 est composée de 5 étapes chacune avec un bloc de convolution et d'identité. Chaque bloc de convolution a 3 couches de convolution et chaque bloc d'identité a également 3 couches de convolution. L'architecture du modèle est affichée dans Figure 5.1. Avant que les couches entièrement ne soient connectées au réseau, nous avons appliqué une mise en commun moyenne globale au modèle de base. La tête du réseau est composée de 2 couches entièrement connectées avec 64 neurones dans la première couche précédées d'un dropout de 0,5 et un neurone par chaque catégorie dans la couche de sortie correspondant à cinq classes différentes ([CBB(Cassava Bacterial Blight), CBSD(Cassava Brown Streak Disease), CGM(Cassava Green Mottle), CMD(Cassava Mosaic Disease), sain(Healthy)]). Les fonctions d'activation utilisées dans les couches convolutionnelles et cachées étaient ReLU (Rectifier Linear Unit) et la couche de sortie était softmax (pour un cas multi-classes) pas sigmoïde (pour un cas binaire) fonction. Le modèle ResNet50 final se compose de 179 couches et 23719173 paramètres au total.

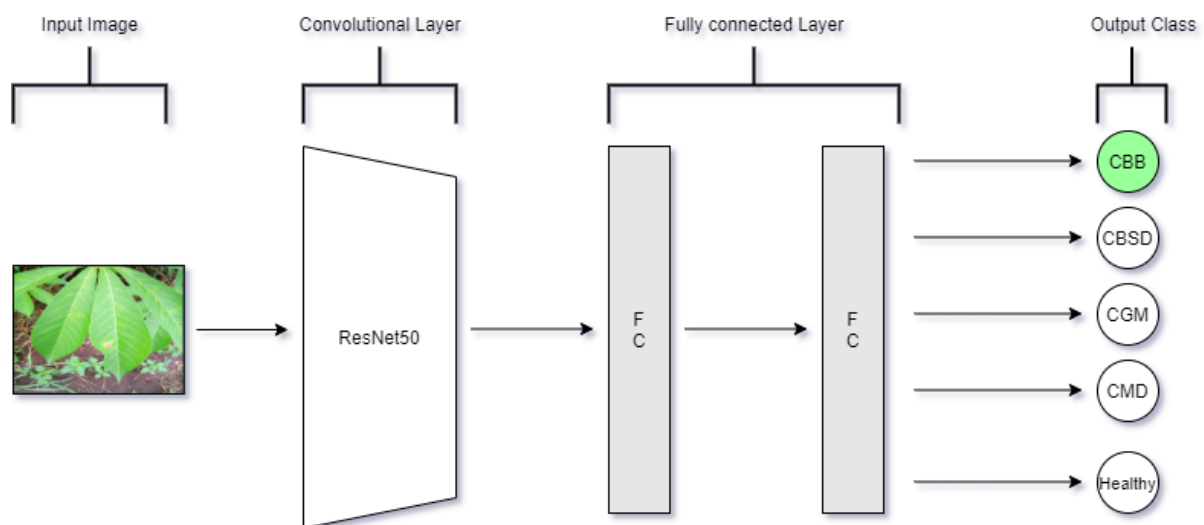


FIGURE 5.1. architecture du modèle ResNet50

### 5.1.2. Modèle EfficientNetB5

L'architecture du modèle EfficientNetB5 est composée de 7 blocs chacune. Ces blocs ont en outre un nombre variable de sous-blocs. L'architecture du modèle est affichée dans Figure 5.2. Avant que les couches entièrement ne soient connectées au réseau, nous avons appliqué une mise en commun moyenne globale au modèle de base. La tête du réseau est composée de 2 couches entièrement connectées avec 64 neurones dans la première couche précédées d'un dropout de 0,5 et un neurone par chaque catégorie dans la couche de sortie correspondant à cinq classes différentes ([CBB(Cassava Bacterial Blight), CBSD(Cassava Brown Streak Disease), CGM(Cassava Green Mottle), CMD(Cassava Mosaic Disease), sain(Healthy)]). Les fonctions d'activation utilisées dans les couches convolutionnelles et cachées étaient ReLU (Rectifier Linear Unit) et la couche de sortie était softmax (pour un cas multi-classes) pas sigmoïde (pour un cas binaire) fonction. Le modèle EfficientNetB5 final se compose de 580 couches et 28644988 paramètres au total.

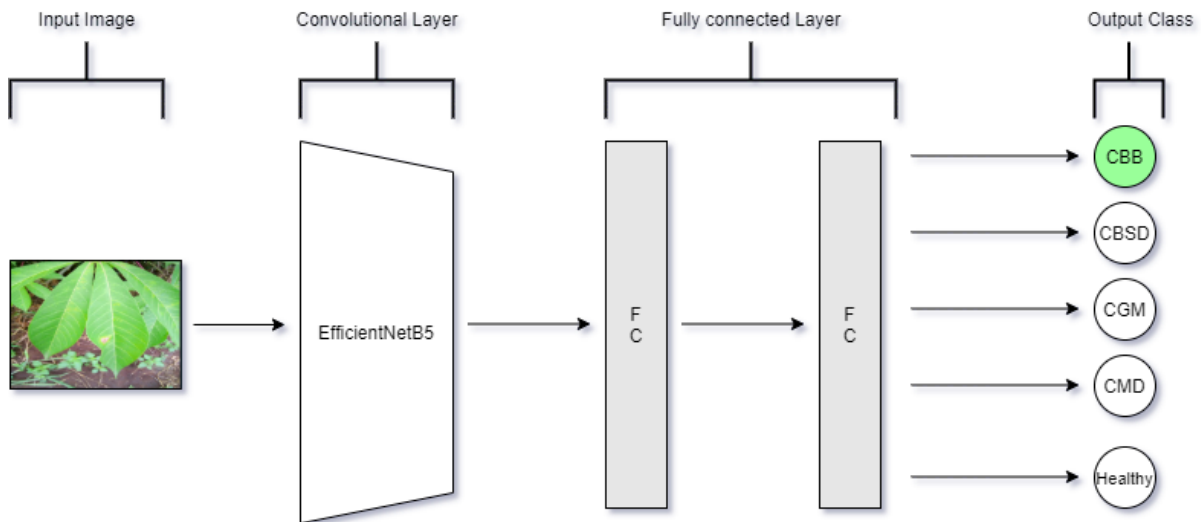


FIGURE 5.2. architecture du modèle EfficientNetB5



## 5.2. Entraînement du modèle

Au cours de la formation, on a eu à tester deux optimiseurs à savoir l'optimiseur SGD et Adam avec différents taux d'apprentissage comme le résume le tableau suivant :

Optimiseurs	Taux D'apprentissage
SGD	0.01
	0.001
	0.0001
ADAM	0.01
	0.001
	0.0001

**TABLE 5.1.** Tableau des Optimiseurs

Ensuite, on a eu à utiliser différentes fonctions au cours de notre apprentissage :

- **ReduceLROnPlateau** : Cette fonction nous permet de réduire notre taux d'apprentissage lorsque celui-ci stagne. Au cours de notre formation, on a utilisé un nombre de 10 d'époques sans amélioration. Si aucune amélioration n'est observée au cours de ce nombre d'époques, le taux d'apprentissage sera réduit par facteur de 2 utilisé pour cette formation.
- **EarlyStopping** : En supposant que notre but est de minimiser notre perte. L'utilisation de cette fonction nous permet de vérifier à la fin de chaque époque si la perte ne diminue plus. Et une fois qu'il ne diminue plus, la formation se termine.
- **ModelCheckpoint** : cette fonction nous a permis d'enregistrer notre meilleur modèle. Au cours de la formation, après chaque époque, il vérifie laquelle a la meilleure précision et l'enregistre.

Puis, on a constaté un déséquilibre au niveau de nos données. Ainsi, on a eu à affecter un poids à chaque classe pour essayer de résoudre ce déséquilibre. Pour cela, on a affecté un poids de 0.1 à la classe majoritaire et pour les autres classes on a procédé par ce calcul suivant : On a calculé le rapport entre la classe dont on calcule le poids et la classe majoritaire et ensuite on a pris ce rapport qu'on a multiplié par le poids de la classe majoritaire pour avoir le poids de la classe dont le poids est calculé.

Enfin, on a utilisé une augmentation sur nos données avec une fonction permettant de donner un lot avec des données transformées lors de notre entraînement. Pour l'augmentation de nos données, on a utilisé beaucoup de transformations sur nos données à savoir :

- **HorizontalFlip** : cette transformation a permis de mettre nos images en horizontal en fixant une probabilité de 0.5 d'avoir celle-ci.
- **VerticalFlip** : cette transformation a permis de mettre nos images en vertical en fixant une probabilité de 0.5 d'avoir celle-ci.
- **ShiftScaleRotate** : cette transformation a permis de mettre nos images en rotation en fixant l'angle de rotation à 45 et aussi en fixant une probabilité de 0.75 d'avoir celle-ci.
- **RandomBrightnessContrast** : cette transformation a permis de rendre nos images plus claires en fixant une probabilité de 0.5 et aussi une limite de la luminosité.
- **RandomCrop** : cette transformation a permis de recadrer nos images en fixant une probabilité de 1.0 et aussi la largeur et la hauteur de notre cadre.

## Résultats et Discussions

Au cours de l'entraînement du modèle, on a eu à tester deux modèles comme indiquée dans la section [architecture du modèle](#) avec deux optimiseurs et aussi avec des fonctions comme l'indique la section [entraînement du modèle](#).

### 1. Modèle Resnet50

L'entraînement de ce modèle a été effectuée en trois phases :

#### ■ Première phase

Dans cette phase, l'entraînement a été faite sans aucune augmentation et sans transformation dans nos données pour mieux voir le meilleur optimiseur et on a sauvegardé nos résultats pour chaque optimiseur avec son taux d'apprentissage. D'abord, on a testé l'optimiseur SGD, on a obtenu les résultats suivants pour chaque taux d'apprentissage comme l'indique le tableau et les figures suivantes :

Optimiseurs	Taux D'apprentissage	Précision(en %)
SGD	0.01	74
	0.001	72
	0.0001	69

**TABLE 6.1.** Effet des différentes taux d'apprentissage de SGD sur la précision du modèle Resnet50.

	precision	recall	f1-score	support		precision	recall	f1-score	support
CBB	0.49	0.38	0.42	217	CBB	0.43	0.39	0.41	217
CBSD	0.62	0.49	0.55	437	CBSD	0.53	0.48	0.51	437
CGM	0.52	0.35	0.42	477	CGM	0.48	0.39	0.43	477
CMD	0.83	0.95	0.89	2631	CMD	0.85	0.91	0.88	2631
HEALTHY	0.51	0.43	0.46	515	HEALTHY	0.41	0.38	0.40	515
accuracy			0.74	4277	accuracy			0.72	4277
macro avg	0.59	0.52	0.55	4277	macro avg	0.54	0.51	0.52	4277
weighted avg	0.72	0.74	0.72	4277	weighted avg	0.70	0.72	0.71	4277

	precision	recall	f1-score	support
CBB	0.42	0.23	0.29	217
CBSD	0.47	0.41	0.44	437
CGM	0.42	0.23	0.30	477
CMD	0.81	0.91	0.86	2631
HEALTHY	0.40	0.43	0.41	515
accuracy			0.69	4277
macro avg	0.50	0.44	0.46	4277
weighted avg	0.66	0.69	0.67	4277

**FIGURE 6.1.** Report de classification des effets des différentes taux d'apprentissage de SGD sur la précision du modèle Resnet50.

Ensuite, on a testé l'optimiseur ADAM et on a obtenu les résultats suivants pour chaque taux d'apprentissage comme le résume le tableau et les figures suivantes :

Optimiseurs	Taux D'apprentissage	Précision(en %)
ADAM	0.01	70
	0.001	72
	0.0001	73

**TABLE 6.2.** Effet des différents taux d'apprentissage de SGD sur la précision du modèle Resnet50.

	precision	recall	f1-score	support		precision	recall	f1-score	support
CBB	0.44	0.42	0.43	217	CBB	0.41	0.26	0.32	217
CBSD	0.51	0.39	0.45	437	CBSD	0.45	0.54	0.49	437
CGM	0.44	0.36	0.39	477	CGM	0.54	0.27	0.36	477
CMD	0.87	0.88	0.87	2631	CMD	0.85	0.91	0.88	2631
HEALTHY	0.38	0.49	0.43	515	HEALTHY	0.43	0.47	0.45	515
accuracy			0.70	4277	accuracy			0.72	4277
macro avg	0.53	0.51	0.51	4277	macro avg	0.54	0.49	0.50	4277
weighted avg	0.70	0.70	0.70	4277	weighted avg	0.70	0.72	0.70	4277

	precision	recall	f1-score	support
CBB	0.41	0.37	0.39	217
CBSD	0.54	0.49	0.51	437
CGM	0.51	0.35	0.42	477
CMD	0.86	0.92	0.89	2631
HEALTHY	0.42	0.43	0.43	515
accuracy			0.73	4277
macro avg	0.55	0.51	0.53	4277
weighted avg	0.71	0.73	0.72	4277

**FIGURE 6.2.** Report de classification des effets des différents taux d'apprentissage de ADAM sur la précision du modèle Resnet50.

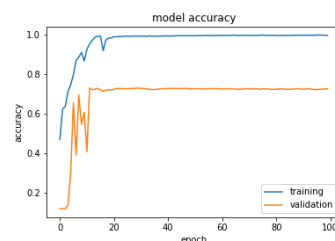
Après entraînement, on a pu voir que SGD avec un taux d'apprentissage de 0.01 a été le meilleur optimiseur avec une précision de 74% comme le montre le tableau 6.1. En effet, on a pu avoir une meilleure précision avec cet optimiseur avec des données sans augmentation et sans transformation. Mais aussi on a noté un déséquilibre au niveau de nos données comme le montre le report de la classification de l'optimiseur SGD avec un taux d'apprentissage de 0.01 où l'on note que avec la classe CMD on a eu une forte précision de 87% qui est supérieure fortement aux autres classes. Pour faire face à ce problème, on a eu à tester des solutions dans les deux phases suivantes avec notre meilleur optimiseur (SGD avec learning rate de 0.01).

## ■ Deuxième phase

Dans cette phase, on a ajouté un tableau de poids avec la variable `class_weight` (voir code) car on a constaté que la classe 3 (CMD) est la classe majoritaire. Dans cette optique on a fixé le poids de cette classe à 0.1 et on a calculé le poids des autres classes en faisant le rapport de ses classes par rapport à la classe majoritaire et ensuite multiplié le résultat par le poids de la classe majoritaire comme le montre équation suivante :

$$W_j = 0.1 * \frac{n_{samples_{majoritaire}}}{n_{sample_j}} \quad (6.1)$$

Après cela, l'entraînement a été faite avec sans augmentation et on a eu les résultats suivants comme l'indique les figures suivantes :



**FIGURE 6.3.** Report de classification de l'effet de la classe weight sur la précision du modèle Resnet50.

On constate que avec l'affectation de poids pour chaque classe, on a vu notre précision diminuer passant de 74% à 73% et aussi toujours un déséquilibre au niveau de nos données montrant nettement la domination de la classe 3(CMD) avec une précision de 87% sur les autres classes. Donc l'affectation de poids au niveau des classes n'a pas eu d'effet sur notre précision car cette dernière a chuté. Donc dans la phase suivante, nous allons effectuer quelques transformations de nos données en appliquant des augmentations sur ces dernières.

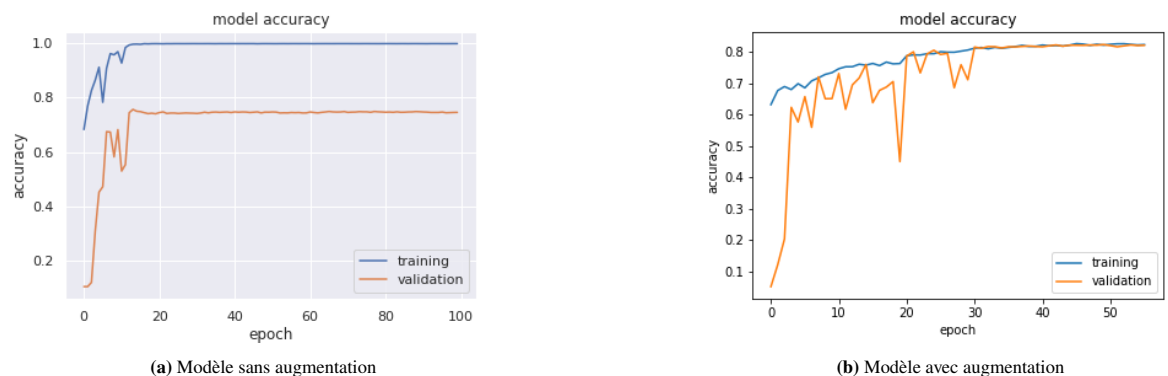
### ■ Troisième phase

Dans cette phase, nous avons ajouté une augmentation au niveau de nos données. Pour l'augmentation, nous avons utilisé les transformations suivantes avec chacune sa précision comme le montre le tableau suivant :

Transformations	Précision(en %)
Horizontal Flip	80
Vertical Flip	77
Shift Scale Rotate	80
Brightness Contrast	81
Crop	81

**TABLE 6.3.** Tableau des différentes transformations de nos données et leur précision pour le modèle ResNet50.

Avec l'augmentation de nos données, nous avons eu une grande augmentation au niveau de notre précision. Nous avons constaté une nette augmentation de notre précision passant de 74% sans augmentation à 81% lors de l'application d'augmentation sur nos données comme le montre les figures suivantes :



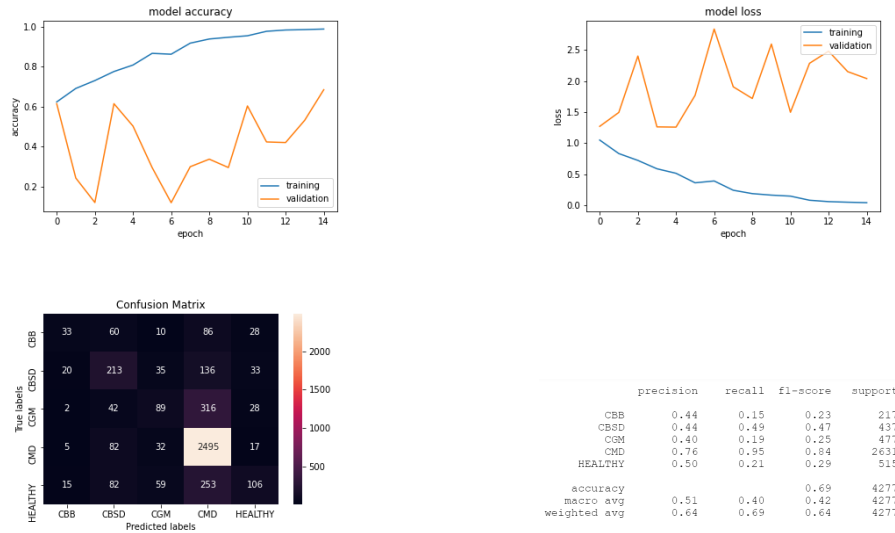
**FIGURE 6.4.** Différence sur la précision du modèle Resnet50 sans augmentation et avec augmentation.

## 2. Modèle EfficientNetB5

Comme avec le modèle précédent, l'entraînement a été effectuée en trois phases :

### (a) Première phase

Dans cette phase , l'entraînement a été faite sans augmentation et sans transformation de nos données mais avec notre meilleur optimiseur SGD avec un taux d'apprentissage de 0.01. Les résultats obtenus ont été les suivantes :



**FIGURE 6.5.** figures montrant la précision sur le modèle EfficientNetB5

Après entraînement, nous avons eu une précision de 69%. Pour augmenter cette précision, nous avons eu à faire tester quelques développées dans les phases suivantes.

### (b) Deuxieme phase

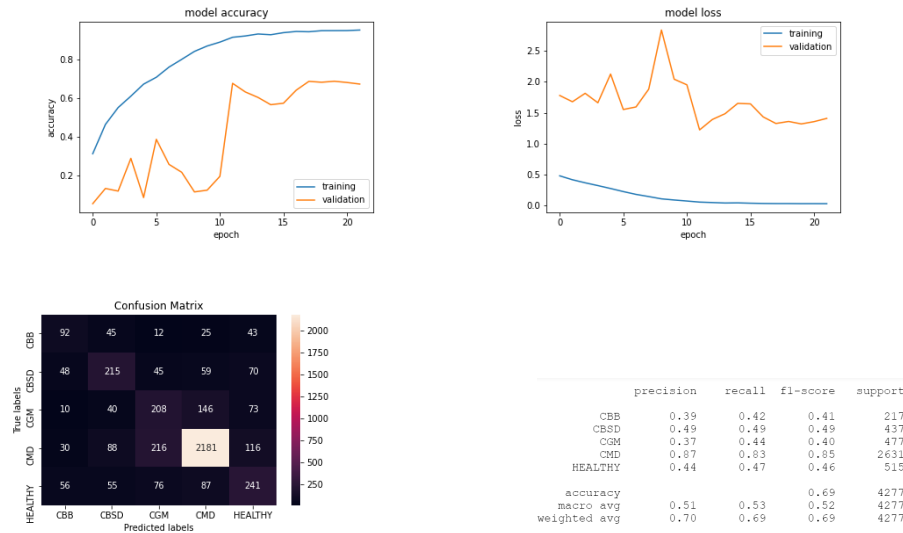
Dans cette phase, on a ajouté un tableau de poids avec la variable `class_weight` (voir code) car on a constaté que la classe 3 (CMD) est la classe majoritaire avec l'utilisation de l'équation 6.1. Et les résultats suivants ont été enregistrés :

On constate que avec l'affectation de poids pour chaque classe, on a vu notre précision n'a pas eu de changement et aussi toujours un déséquilibre au niveau de nos données montrant nettement la domination de la classe 3 (CMD) avec une précision de 87% sur les autres classes. Donc l'affectation de poids au niveau des classes n'a pas eu d'effet sur notre précision car cette dernière a chuté. Donc dans la phase suivante, nous allons effectuer quelques transformations de nos données en appliquant des augmentations.

### (c) Troisième phase

Dans cette phase, nous avons appliqué les mêmes augmentations au niveau de nos données comme le modèle précédent et nous avons eu les résultats suivantes :



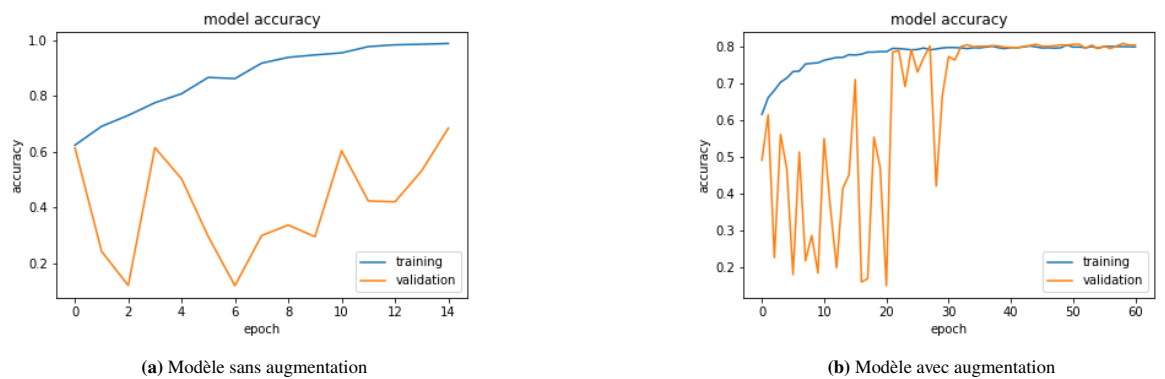


**FIGURE 6.6.** Effets des poids sur la précision sur le modèle EfficientNetB5

Transformations	Précision(en %)
Horizontal Flip	76
Vertical Flip	79
Shift Scale Rotate	81
Brightness Contrast	81
Crop	80

**TABLE 6.4.** Tableau des différentes transformations de nos données et leur précision pour le modèle EfficientNetB5.

Avec l'augmentation de nos données, nous avons eu une grande augmentation au niveau de notre précision. Nous avons constaté une nette augmentation de notre précision passant de 69% sans augmentation à 80% lors de l'application d'augmentation sur nos données comme le montre les figures suivantes :



**FIGURE 6.7.** Différence sur la précision du modèle EfficientNetB5 sans augmentation et avec augmentation.

On a pu tester le modèle ResNet 50 avec de différentes méthodes à savoir sans augmentation de données, avec `class_weight` et avec augmentation de données et de même que le modèle EfficientNetB5. Et cela nous a montré que avec le modele Restnet50 qu'avec l'augmentation de données on a pu avoir une meilleure précision de 81% mais aussi on a noté une diminution de notre précision lorsqu'on a créé de poids pour chaque lors de l'entraînement. Et aussi pour le modèle EfficientNetB5 , nous avons eu lors de notre résultat une précision de 80% et aussi aucun changement de la précision avec l'utilisation des poids de chaque classe comme qu'avec l'utilisation de nos données sans transformation et augmentation.

## Conclusions

Dans ce travail, nous avons mise en place des modèles d'apprentissage profond (Resnet50 et EfficientNetB5) pour détecter automatiquement les maladies présentes dans les feuilles manioc. Les modèles ont été formés avec un ensemble de données réparties en 4 classes de maladies et une classe comportant les feuilles saines. Nous avons constaté que les données étaient fortement biaisées vers la classe cmd et par conséquent nous avons eu à appliquer une technique pour contrarier ce problème. Notre objectif était de trouver un modèle capable de bien faire cette tâche. En effet, les résultats ont été meilleurs car nous avons eu une bonne précision avec nos deux modèles. Nous avons utilisé plusieurs techniques permettant d'améliorer notre précision et aussi d'avoir de meilleurs résultats parmi ses techniques on peut en citer en autre : le poids de classe, augmentation de nos données avec l'utilisation de la bibliothèque *Augmentations* de Keras offrant une api propre et facile à utiliser. D'après nos résultats, nous avons constaté que le poids de classe n'a pas eu d'effet car nos données ont été toujours biaisées vers la classe cmd et aussi l'augmentation de nos données à participer fortement à l'obtention d'une meilleure précision passant de 74% à 81% pour le modèle Resnet50 et de 69% à 80% pour le modèle EfficientNetB5. Nous prévoyons que notre système proposé apportera une contribution significative dans le domaine de la recherche agricole. Mais Le présent projet n'étant qu'une étape pour la résolution de ce fléau, plusieurs recommandations peuvent être proposées pour rendre ce dernier plus sophistiqué.

La première recommandation est d'augmenter le nombre d'images dans la base de données pour rendre le modèle plus performant. Vu que nous utilisons nos propre ordinateur il sera impossible de recueillir un très grand nombre de données. Ceci pourrait être géré par une société technologique capable de manipuler une très grande quantité de données. Ces données sont important pour l'amélioration de ce projet car un très grand nombre de données favorisera le taux de précision. Comme nous l'avons fait dans les travaux précédents avec l'augmentation des données qui à augmenter considérablement le pourcentage de précision.

La deuxième recommandation est de tester dans le futur plusieurs autres modèles afin d'en déduire le mieux adapté et de comparer leur pourcentage de précision ainsi que leur pourcentage d'erreur et aussi prévoir dans l'avenir d'intégrer ResNet50 et EfficientNetB5 dans un même modèle et aussi d'essayer d'autres techniques pour contrer la classe biaisée et d'obtenir des données équilibrées dans chaque classe. Dans ce cas la classification des images sera plus précise car selon le modèle et les données recueillies peuvent augmenter le pourcentage de ces deux indices. Par la suite il serait intéressant d'intégrer notre modèle dans des applications mobiles telles les smartphones afin d'avoir accès aux informations relatives à leur culture (savoir si cette feuille est contaminée, si une feuille est en bonne santé etc.) vu la situation économique en Afrique. Et aussi de se baser sur ce projet pour lier à l'apprentissage automatique pour en développer d'autres avec des cultures différentes.

# Références

- [1] <https://www.hebergementwebs.com/nouvelles/pretraitement-des-donnees-6-etapes-necessaires-pour-les-data-scientists>.
- [2] <https://ichi.pro/fr/pretraitement-des-donnees-avec-python-62761617448240>.
- [3] <https://www.quantmetry.com/blog/data-augmentation-image/>.
- [4] Aduni Ufuan ACHIDI et al. « The Use of Cassava Leaves as Food in Africa ». In : *Ecology of Food and Nutrition* 44.6 (nov. 2005). Publisher : Routledge \_eprint : <https://doi.org/10.1080/03670240500348771>, p. 423-435. ISSN : 0367-0244. DOI : [10.1080/03670240500348771](https://doi.org/10.1080/03670240500348771). URL : <https://doi.org/10.1080/03670240500348771> (visité le 24/02/2021).
- [5] Konstantinos P. FERENTINOS. « Deep learning models for plant disease detection and diagnosis ». In : *Computers and Electronics in Agriculture* 145 (1<sup>er</sup> fév. 2018), p. 311-318. ISSN : 0168-1699. DOI : [10.1016/j.compag.2018.01.009](https://doi.org/10.1016/j.compag.2018.01.009). URL : <https://www.sciencedirect.com/science/article/pii/S0168169917311742> (visité le 21/04/2021).
- [6] Alvaro FUENTES et al. « A Robust Deep-Learning-Based Detector for Real-Time Tomato Plant Diseases and Pests Recognition ». In : *Sensors* 17.9 (sept. 2017). Number : 9 Publisher : Multidisciplinary Digital Publishing Institute, p. 2022. DOI : [10.3390/s17092022](https://doi.org/10.3390/s17092022). URL : <https://www.mdpi.com/1424-8220/17/9/2022> (visité le 21/04/2021).
- [7] R J HILLOCKS et J M THRESH. « CASSAVA MOSAIC AND CASSAVA BROWN STREAK VIRUS DISEASES IN AFRICA : » en. In : (2000), p. 8.
- [8] G. W. OTIMNAPE, T. ALICAI et J. M. THRESH. « Changes in the incidence and severity of Cassava mosaic virus disease, varietal diversity and cassava production in Uganda ». en. In : *Annals of Applied Biology* 138.3 (2001). \_eprint : <https://onlinelibrary.wiley.com/doi/pdf/10.1111/j.1744-7348.2001.tb00116.x>, p. 313-327. ISSN : 1744-7348. DOI : <https://doi.org/10.1111/j.1744-7348.2001.tb00116.x>. URL : <https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1744-7348.2001.tb00116.x> (visité le 24/02/2021).
- [9] Godliver OWOMUGISHA. « Computational intelligence & modeling of crop disease data in Africa ». ISBN : 9789403426365. Thèse de doct. University of Groningen, 28 août 2020. DOI : [10.33612/diss.130773079](https://doi.org/10.33612/diss.130773079). URL : <http://hdl.handle.net/11370/d34c8a5f-d341-4d8c-aa47-88ada16967b5> (visité le 20/04/2021).
- [10] Amanda RAMCHARAN et al. « Deep Learning for Image-Based Cassava Disease Detection ». In : *Frontiers in Plant Science* 8 (2017). Publisher : Frontiers. ISSN : 1664-462X. DOI : [10.3389/fpls.2017.01852](https://doi.org/10.3389/fpls.2017.01852). URL : <https://www.frontiersin.org/articles/10.3389/fpls.2017.01852/full?report=reader> (visité le 21/04/2021).
- [11] Vijai SINGH et A. K. MISRA. « Detection of plant leaf diseases using image segmentation and soft computing techniques ». en. In : *Information Processing in Agriculture* 4.1 (mar. 2017), p. 41-49. ISSN : 2214-3173. DOI : [10.1016/j.inpa.2016.10.005](https://doi.org/10.1016/j.inpa.2016.10.005). URL : <https://www.sciencedirect.com/science/article/pii/S2214317316300154> (visité le 24/02/2021).
- [12] Kehinde A. TAIWO. « Utilization Potentials of Cassava in Nigeria : The Domestic and Industrial Products ». In : *Food Reviews International* 22.1 (jan. 2006). Publisher : Taylor & Francis \_eprint : <https://doi.org/10.1080/87559120500379787>, p. 29-42. ISSN : 8755-9129. DOI : [10.1080/87559120500379787](https://doi.org/10.1080/87559120500379787). URL : <https://doi.org/10.1080/87559120500379787> (visité le 24/02/2021).