

---

# RECONNAISSANCE DES PLANTES

---

Membres de l'équipe projet : Bernadette GASMI, Jean-Baptiste MACK, Morgan PERCHEC, Lionel SCHEIDER

6 novembre 2025



Jean-Baptiste MACK



Morgan PERCHEC



Bernadette GASMI



Lionel SCHNEIDER



Résumé : Dans le cadre de notre formation DataScientist, nous mettons en pratiques les techniques d'intelligence artificielle enseignées avec un passage obligé par le Machine Learning pour progressivement conclure vers le Deep Learning.

Mots clés : Machine Learning, Deep Learning, Classification, Métriques

## Table des matières

1.	Introduction .....	6
1.1.	<b>Cadrage du projet.....</b>	<b>6</b>
1.1.1.	Contexte et enjeux du projet.....	6
1.1.2.	Gestion du projet.....	6
1.1.3.	Scénario .....	8
1.1.4.	Objectifs.....	8
1.1.5.	Etat des lieux .....	9
1.1.6.	Revue littéraire .....	9
1.1.7.	Présentation des datasets .....	10
2.	Compréhension et manipulation des données .....	11
2.1.	Exploration des données dataset PlantVillage/Segmented .....	11
3.	Modélisation .....	15
3.1.	Machine Learning (sans DL).....	15
3.1.1.	<b>Méthodologie .....</b>	<b>15</b>
3.1.2.	<b>Extraire les caractéristiques.....</b>	<b>16</b>
3.1.3.	<b>Explorer les caractéristiques extraites.....</b>	<b>17</b>
3.1.4.	<b>Ré-échantillonnage.....</b>	<b>19</b>
3.1.5.	<b>Split train/test (et éventuellement validation) .....</b>	<b>19</b>
3.1.6.	<b>Pré-traitements .....</b>	<b>19</b>
3.1.6.1.	<b>Data augmentation.....</b>	<b>19</b>
3.1.6.2.	<b>Standardisation / normalisation des features .....</b>	<b>20</b>

3.1.7.	<b>Sélection des meilleures caractéristiques .....</b>	21
3.1.8.	<b>Entrainement des modèles et évaluations .....</b>	22
3.2.	<b>Deep Learning .....</b>	25
3.2.1.	<b>Méthodologie .....</b>	25
3.2.2.	<b>Définition des critères de sélection .....</b>	25
3.2.3.	<b>Exploration d'architectures DL .....</b>	26
3.2.4.	<b>Évaluation comparative .....</b>	34
3.2.5.	<b>Sélection et recommandations.....</b>	36
3.2.6.	<b>Limites et perspectives .....</b>	38
4.	<b>Conclusion .....</b>	38
5.	<b>Bibliographie .....</b>	39
6.	<b>Annexes .....</b>	40
6.1.	<b>Analyse exploratoire des caractéristiques extraites .....</b>	40
6.2.	<b>Les méthodes pour analyser l'importance des features.....</b>	49
6.3.	<b>Comparaison des caractéristiques des modèles ML.....</b>	50
6.4.	<b>Métriques trackées des architectures DL .....</b>	50
6.5.	<b>Caractéristiques des features .....</b>	53

## Table des illustrations

Figure 1: Environnement de travail .....	7
Figure 2 : Sélection du dataset pour premier cycle .....	10
Figure 3 : extrait des images du dataset Plantvillage/segmented pour les 14 types de plantes .....	11
Figure 4 : Nombre d'images par espèce.....	12
Figure 5 : Distribution des classes maladie .....	13
Figure 6 : Répartition des images saines vs malades par espèce, Distribution des classes saines, Distribution des classes malades .....	13
Figure 7 : Schéma de principe des activités à mener pour ML (sans DL) .....	15
Figure 8 : Caractéristiques extraites des feuilles de plantes.....	16
Figure 9 : Nouvelle répartition avec la data augmentation.....	20
Figure 10 : Importance globale des variables (SHAP) — Top 25 .....	21
Figure 11: Importance des features par classe (SHAP) — Top-5 classes .....	22
Figure 12 : Performances des modèles ML par classe .....	24
Figure 13 : comparaison et corrélation Accuracy et Macro F1-Score par architecture - cas 1 .....	34
Figure 15 : comparaison et corrélation Accuracy et Macro F1-Score par architecture - cas 2 .....	35
Figure 17 : comparaison et corrélation Accuracy et Macro F1-Score par architecture - cas 3 .....	36

Tableau 1 : Comparatif des datasets .....	11
Tableau 2 : Performances des modèles ML sur l'ensemble de test .....	23
Tableau 3 : Critères de sélection des architectures .....	26
Tableau 4 : Comparatif des caractéristiques des modèles pré-entraînés choisis .....	27
Tableau 5 : Synthèse des performances des 9 architectures .....	33
Tableau 6 : classement final des 6 architectures pour l'identification de l'espèce.....	35
Tableau 7 : classement final des 6 architectures pour l'identification de la maladie .....	35
Tableau 8 : classement final des 6 architectures pour l'identification complète .....	36
Tableau : Synthèse des critères évalués pour les architectures retenues .....	37
Tableau : synthèse des performances obtenues par cas .....	37

## **Acronymes et sigles**

Ce tableau présente les acronymes et sigles employés dans ce rapport.

Acronyme	Signification
ML	Machine Learning
DL	Deep Learning
IA	Intelligence Artificielle

# 1. Introduction

## 1.1. Cadrage du projet

### 1.1.1. Contexte et enjeux du projet

Les plantes occupent une place essentielle dans notre vie quotidienne, que ce soit pour alimenter les populations, protéger la biodiversité ou embellir nos paysages. Pourtant, leur santé est régulièrement menacée par des maladies, des parasites ou des conditions climatiques défavorables, entraînant des pertes économiques colossales — estimées à plusieurs centaines de milliards de dollars par an selon la FAO — et compromettant la sécurité alimentaire mondiale. En parallèle, l'engouement du grand public pour la reconnaissance des espèces et la compréhension de leur état de santé ne cesse de croître, comme le montre l'adoption massive d'outils collaboratifs tels que PlantNet.

Ce projet s'inscrit dans cette dynamique, développé dans le cadre d'une formation en data science. Il consiste à concevoir une solution basée sur des techniques d'intelligence artificielle pour identifier automatiquement les espèces végétales, évaluer leur santé et diagnostiquer d'éventuelles maladies. À travers ce travail, nous mettons en pratique des méthodes avancées d'apprentissage automatique tout en nous confrontant à des enjeux réels. Il illustre ainsi le potentiel de la data science pour répondre à des défis sociétaux, tout en renforçant nos compétences pour imaginer des solutions techniques innovantes.

Nous sommes trois collaborateurs sans expertise dans ce domaine et un architecte IA qui possède des compétences dans l'industrialisation du Deep Learning.

### 1.1.2. Gestion du projet

Les points clé de la gestion du projet :

Réunions : Hebdomadaire et au besoin.

Pratique : Tout au long du projet, pour les étapes d'analyse exploratoire des datasets, de modélisation ML et DL, nous avons favorisé : une phase d'exploration informelle pour chaque membre de l'équipe et une phase de synthèse élaborée par l'équipe. C'est le meilleur moyen de s'assurer de la mise en application de notre formation mais également de l'émergence des idées.

Environnement de travail :

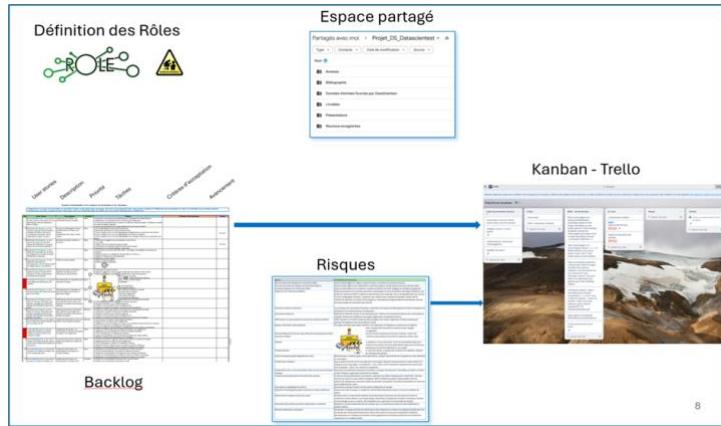


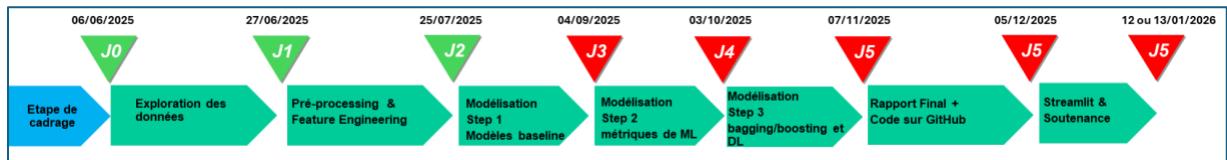
Figure 1: Environnement de travail

### Environnement de développement

La gestion de configuration s'effectue avec Github. Nous avons créé un environnement nommé `conda_env`.

### Cycle de vie du projet

Le cycle de vie du projet est le suivant :



L'avancement du projet s'appuie sur les décisions de chaque jalon.

### Risques projet

#### Risque 1 : Beaucoup d'itérations

- Cause : Le démarrage du projet arrive trop tôt par rapport au planning des apprentissages nécessaires pour le projet.
- Mitigation : Lire les modules des cours par anticipation

#### Risque 2 : Des jalons non tenus

- Cause : La performance de nos PC
- Mitigation : VM de DataScientest, Google Colab, Changer de PC

#### Risque 3 : Erreurs probables sur les interprétations

- Cause : La durée entre jalons très courte
- Mitigation : Suites aux analyses exploratoires, les recommandations ne sont pas toutes appliquées mais elles ne sont pas perdues et elles seront considérées dans les étapes suivantes suivant l'état d'avancement.

### 1.1.3. Scénario

La figure suivante illustre le scénario global établi à partir de la mission du projet.

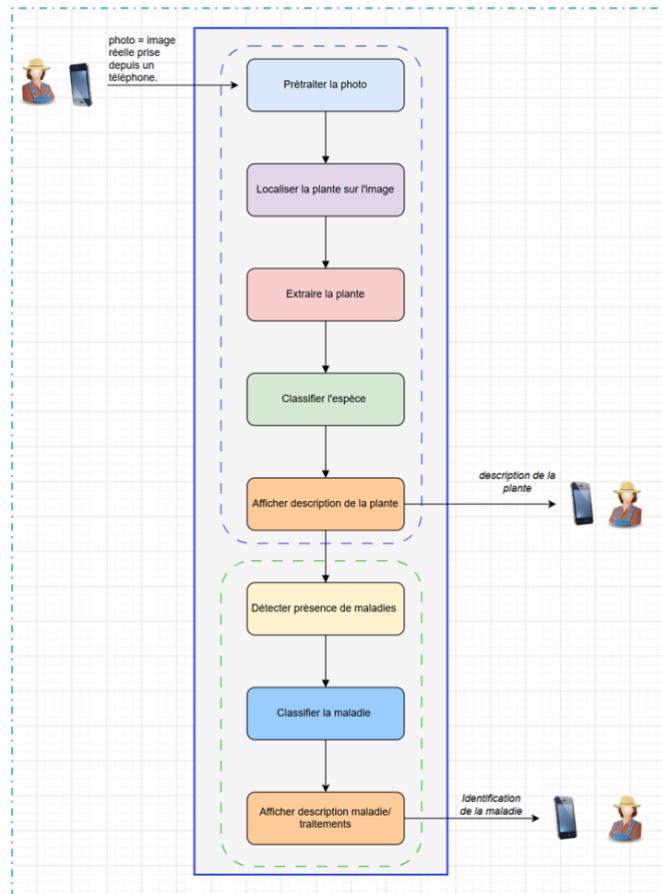


Figure 1 : Scénario global

Lors de la revue du jalon1, parmi les scénarios envisagés, « Détection de plantes invasives ou nuisibles parmi des plantules de maïs », « Surveillance de la croissance de plantes de maïs », « Identifier une espèce », « identifier la maladie de la plante », la décision a été prise de retenir le scénario suivant :

Scénario : A partir d'une photo d'une feuille de plante **cadrée**

- Identifier la classe/espèce de la plante
- Dire si la plante est malade ou saine
- Si elle est malade, donner le nom de la maladie

Les autres étapes seront développées ultérieurement suivant notre avancement.

### 1.1.4. Objectifs

Le scénario nous amène à définir les 3 objectifs suivants :

**Objectif 1 – Classification de l'espèce – Quelle est cette plante ?**

Développer un modèle (multi-classe) capable de reconnaître automatiquement l'espèce de chaque feuille (orange, maïs, tomates, etc.) à partir de ses caractéristiques de forme, couleur et texture...

### **Objectif 2 – Détection de l'état de santé – La plante est-elle malade ?**

Mettre en place un classifieur binaire pour distinguer les feuilles saines de celles présentant des symptômes de maladie.

### **Objectif 3 – Identification du type de maladie – Quelle est la maladie de la plante ?**

Pour les feuilles diagnostiquées « malade », développer un modèle (multi-classe) déterminant précisément la maladie (tavelure, mildiou, rouille, etc.)

Notre projet se situe donc dans un contexte d'apprentissage supervisé. Nos variables cibles sont : le nom de la plante, l'état de santé et le nom de la maladie. Les variables explicatives seront les caractéristiques des feuilles (forme, couleur, contour...).

#### **1.1.5. Etat des lieux**

Nous avons effectué un état des lieux des applications similaires à l'objectif proposé par DataScientest, les datasets et les critères de qualité.

#### **1.1.6. Revue littéraire**

Nous avons parcouru la publication proposée par DataScientest et exploré d'autres. Celle qui a retenu notre attention est une revue systématique des méthodes les plus performantes de Machine Learning, de traitement d'image, et d'algorithme de Deep Learning appliqués à la reconnaissance des espèces végétales, avec citation des datasets utilisés, publiée en 2024.

[https://www.researchgate.net/publication/384455442 A systematic review of deep learning techniques for plant diseases :](https://www.researchgate.net/publication/384455442_A_systematic_review_of_deep_learning_techniques_for_plant_diseases)

Cet article confirme le Deep Learning comme technique la plus performante. Cependant, lors du passage de jalon1, il a été convenu d'effectuer tout le cycle de vie du projet (étapes Machine Learning, Deep Learning) dans un esprit de mise en application des techniques apprises dans le cursus DataScientist.

La revue littéraire et l'état des lieux nous ont permis :

- de comprendre les enjeux, de définir les objectifs du projet dans le détail, avec précision et les risques compte tenu de la complexité du sujet,
- de choisir les datasets qui seront utilisés compte tenu de leurs caractéristiques
- d'envisager des pré-sélections de modèles de Machine Learning et Deep Learning

Des recherches complémentaires ont été effectuées sur Internet afin de mieux comprendre les méthodes de classement, d'organisation et d'archivage de documents. Des ressources variées, comme des sites éducatifs, des forums spécialisés ou encore des blogs techniques, ont été consultées pour enrichir nos connaissances théoriques et pratiques. Par ailleurs, des outils d'intelligence artificielle tels que ChatGPT et Le chat ont été utilisés pour poser des

questions précises, obtenir des explications claires, explorer différentes approches et affiner certaines idées. Ces échanges ont permis de gagner du temps, de mieux structurer les étapes du projet et de répondre efficacement aux obstacles rencontrés.

### 1.1.7. Présentation des datasets

A partir des 6 datasets proposés par DataScientest, nous avons effectué plusieurs sélections successives basées sur des explorations afin de n'en retenir qu'un, PlantVillage, plus simple pour se focaliser sur le cœur du sujet.

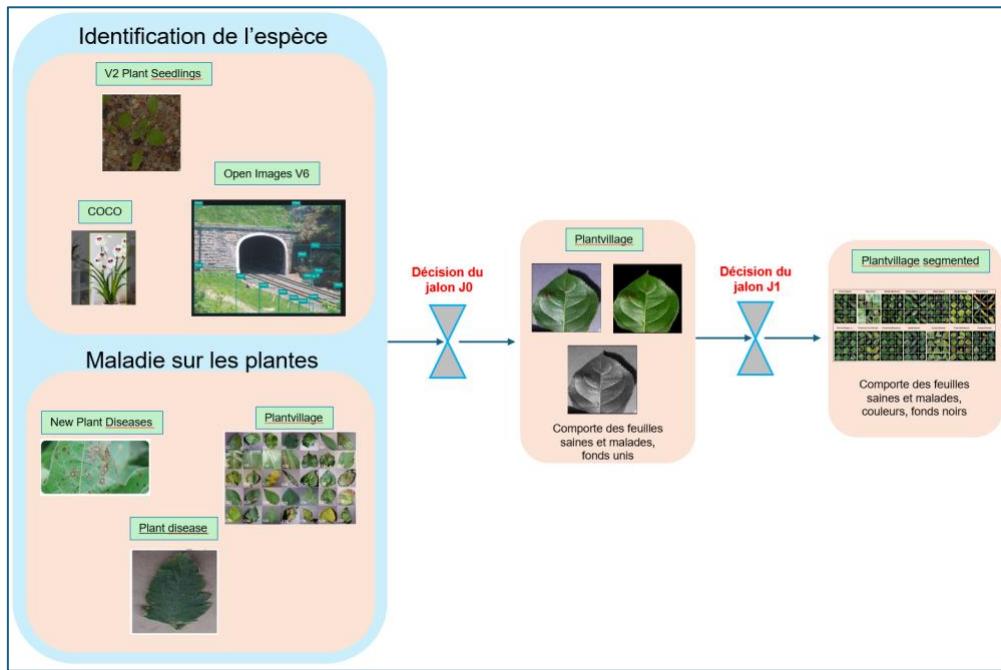


Figure 2 : Sélection du dataset pour premier cycle

Les 3 datasets V2 Plant Seedlings, Open Images V6 et COCO sont éliminés car le scénario 3 part d'une photo feuille cadrée sur fond uni. On trouve en effet des images de plantes avec des environnements différents et peu d'espèces communes entre ces 3 datasets. Le tableau suivant compare les 3 autres datasets permettant la détection des maladies.

Caractéristique	PlantVillage	Plant Disease	New Plant Diseases
Taille du dataset	~54,000 images 38 classes (espèce + maladie ou healthy)	~54,000 images	~88,000 images
Nombre d'espèces	14 espèces de plantes	38 classes de maladies sur plusieurs espèces	> 60 espèces de plantes
Nombre de maladies	26 maladies (avec healthy classes)	>50 maladies (certaines plantes avec plusieurs maladies)	~120 maladies (certaines classes rares)
Type d'image	Images avec des fonds unis	Images avec des fonds unis	Images avec des fonds unis
Annotation	Espèce + maladie + healthy	Espèce + maladie	Espèce + maladie + conditions environnementales
Format	JPG / PNG	JPG	JPG / JPEG / PNG
Accessibilité	Ouvert (Kaggle, GitHub)	Ouvert (Kaggle)	Ouvert (Google Dataset Search, Zenodo, etc.)

Tableau 1 : Comparatif des datasets

Plant Disease est éliminé car pour un même nombre d'images, il fournit plus de types de maladies. Cela n'apporte rien à notre scénario. New Plant Disease est créé à l'aide d'une augmentation hors ligne de PlantVillage. Il comporte 34000 images supplémentaires. Notre analyse exploratoire montre que certaines espèces ont été augmentées plus que d'autres pour couvrir un objectif dédié non précisé dans la littérature. Notre choix se porte sur PlantVillage qui cadre bien à notre scénario. Sa structure est détaillée dans le chapitre suivant. On pourra toujours utiliser les autres datasets lorsque l'architecture du code sera établie et vérifiée et en fonction des résultats obtenus.

## 2. Compréhension et manipulation des données

### 2.1. Exploration des données dataset PlantVillage/Segmented

Le dataset PlantVillage est structuré en 3 dossiers : color, grayscale et segmented. Chaque dossier comporte 54306 images, réparties dans 38 sous-dossiers (les 38 classes). Le nom des sous-dossiers correspond à : "Nom de la plante\_Healthy" ou "Nom de la plante\_Nom de maladie". Compte tenu de notre scénario, nous utiliserons le dataset PlantVillage/Segmented. Il comporte 54306 images, réparties dans 38 sous-dossiers (les 38 classes). Il y a 14 espèces de plantes et 20 classes maladies. La taille du dossier est de 593 Mo.

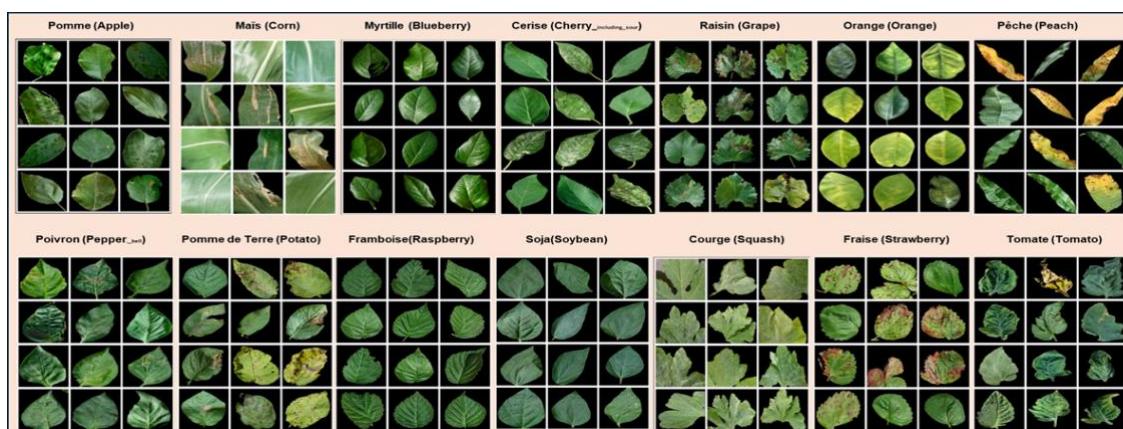


Figure 3 : extrait des images du dataset Plantvillage/segmented pour les 14 types de plantes

## Suppression des images inexploitables

Nous avons détecté et supprimé 18 images quasi noires (dont une complètement noire).

## Détection de doublons d'image et suppression

Nous avons détecté et supprimé 21 doublons.

## Redimensionnement uniforme des images

Sur les 54267 images, la taille la plus fréquente est 256x 256 et il n'y a que 4 images de tailles différentes : [((256, 256), 54263), ((466, 512), 1), ((324, 512), 1), ((335, 512), 1), ((470, 512), 1)]. Elles ont été redimensionnées à 256 x 256.

## Modification des extensions

L'extension .jpg a un nombre très élevé d'images, soit 54302 images et les extensions .jpeg et .png ont chacune seulement 2 images. Leur extension sera modifiée.

## Analyse statistique du dataset PlantVillage/Segmented

### Nombre d'images par espèce et distribution des classes maladie :

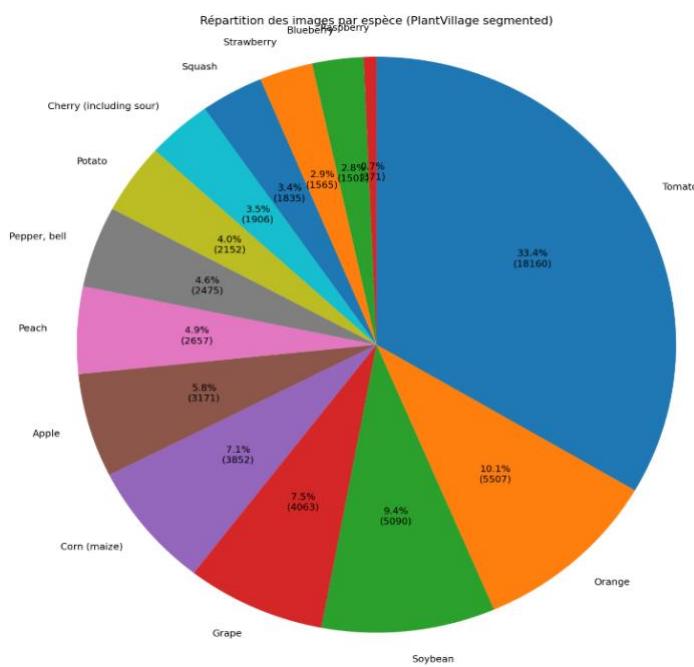
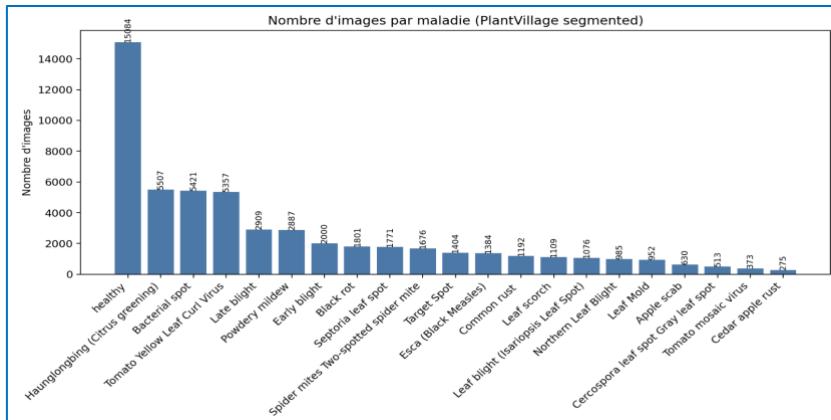


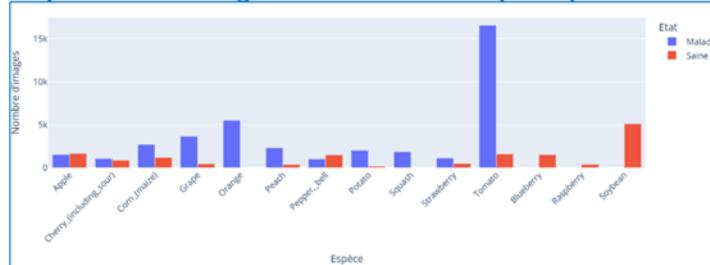
Figure 4 : Nombre d'images par espèce



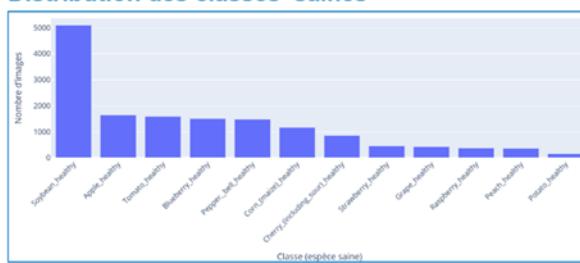
*Figure 5 : Distribution des classes maladie*

#### Répartition des images saines vs malades par espèce, Distribution des classes saines, Distribution des classes malades

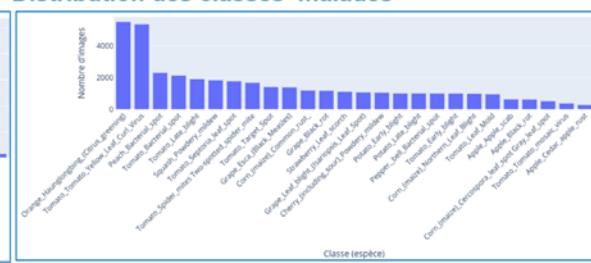
**Répartition des images saines vs malades par espèce**



**Distribution des classes saines"**



**Distribution des classes malades"**



*Figure 6 : Répartition des images saines vs malades par espèce, Distribution des classes saines, Distribution des classes malades*

#### Observations :

- Le dataset montre un très fort déséquilibre entre espèces : Tomato domine avec près de 18 000 images, tandis que Raspberry n'en compte qu'environ 300. Orange et Grape figurent aussi parmi les plus abondantes, alors que Blueberry ou Peach restent peu représentées.
- Au sein de certaines espèces (Tomato, Orange, Grape...), les images malades sont majoritaires, alors que pour d'autres (Pepper\_bell, Potato, Squash...) les exemples sains l'emportent. Les maladies courantes comme Orange\_Huanglongbing ou Tomato\_Yellow\_Leaf\_Curl\_Virus totalisent chacune près de 5 000 images.
- En revanche, de nombreuses maladies rares forment une longue traîne avec moins de 300–500 images.

## Risques :

Risques liés au déséquilibre des données

- Sur-apprentissage sur les espèces majoritaires : La surreprésentation de certaines espèces (ex. Tomato) expose le modèle à un biais vers ces dernières, au détriment des espèces rares (ex. Raspberry), dont la diversité d'exemples est insuffisante pour capturer leurs caractéristiques visuelles spécifiques.
- Biais des modèles binaires globaux : Un modèle ignorant les espèces sera systématiquement influencé par la majorité d'images de chaque espèce, faussant les décisions et rendant impossible la fixation d'un seuil de classification unique et fiable.
- Prédominance des classes majoritaires : Le modèle tendra à privilégier les classes les plus représentées, négligeant les catégories minoritaires et augmentant les erreurs de classification.

Risques liés à l'intégration de l'espèce et de l'état de santé dans le classifieur "maladie" :

Biais croisé espèce/santé → maladie : Le modèle peut "apprendre par raccourci" que certaines maladies collent à certaines espèces/états (ex. Tomato souvent malade, Pepper souvent sain) et s'appuyer davantage sur ces métainformations que sur les indices visuels des lésions. Risque de mauvaise généralisation hors distribution.

Seuils et calibrations non spécifiques à l'espèce : Un classifieur maladie global partage des frontières de décision uniques alors que les distributions varient fortement selon l'espèce et la prévalence "healthy vs diseased". Sans calibrations/thresholds par espèce, on observe des seuils inadaptés (trop laxistes pour certaines espèces, trop stricts pour d'autres).

Ignorance des maladies rares : Les maladies fréquentes dominent l'apprentissage et la décision ; les catégories rares (souvent spécifiques à une espèce) voient leur rappel chuter, augmentant faux négatifs et faux positifs (confusion vers 3–5 maladies majeures).

Propagation d'erreurs et décalage entraînement/inférence : Si l'espèce/santé est prédite en amont et servie en entrée, toute erreur se répercute sur la maladie.

Masquage des combinaisons espèce–maladie : Masque "dur" (n'autoriser que les maladies plausibles pour l'espèce) : limite les erreurs absurdes, mais peut rejeter des cas atypiques/croisés. Pas de masque : autorise des prédictions impossibles si l'espèce est mal prédite, augmentant les faux positifs. Mauvaise séparation "healthy" vs maladies

Si "healthy" est intégré à la tête maladie (classe supplémentaire) ou si l'état de santé est injecté comme feature, le fort déséquilibre sain/malade par espèce peut déplacer les frontières et dégrader la détection de maladies peu fréquentes ou peu contrastées.

## Recommandations :

Les mesures suivantes visent à corriger les biais, améliorer la généralisation du modèle, et garantir une meilleure détection des classes minoritaires :

- Split stratifié : Privilégier un split stratifié afin de préserver les proportions réelles des espèces et des classes saines/malades.

- Rééchantillonnage ciblé : Appliquer un sous-échantillonnage des classes surreprésentées (ex. Tomato\_malade) et un sur-échantillonnage ou une augmentation de données pour les espèces et classes sous-représentées (ex. Potato\_healthy).
- Pondération des classes : Utiliser des class weights équilibrés pour renforcer la prise en compte des catégories rares et corriger les biais liés aux déséquilibres.
- Approche hiérarchique ou multi-étapes : Adopter une classification hiérarchique (ex. binaire par espèce, puis multiclasse pour les maladies) ou intégrer nom\_plante en entrée pour affiner la performance et mieux gérer les déséquilibres.
- Métriques adaptées : f1-score par classe, recall, confusion matrix plutôt qu'accuracy brute.

### Extraction des labels

Pour couvrir les 3 objectifs du projet, nous avons identifié les cibles suivantes : nom\_plante, est\_Saine, nom\_maladie.

## 3. Modélisation

Sachant que le DL offre les meilleurs résultats, nous n'avons pas poussé les explorations du ML, ni effectué de trades pour retenir la meilleure solution, les réservant pour le DL. Par abus de langage, nous utiliserons dans la suite du rapport ML pour ML sans DL.

### 3.1. Machine Learning (sans DL)

#### 3.1.1. Méthodologie

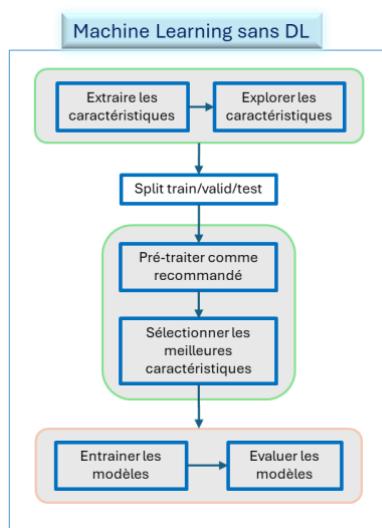


Figure 7 : Schéma de principe des activités à mener pour ML (sans DL)

**Machine Learning :** En apprentissage automatique, les modèles ne traitent pas directement les pixels bruts d'une image. Il faut extraire des caractéristiques numériques, pour les utiliser comme entrées et effectuer une exploration statistique. Un rééchantillonnage (sur le train uniquement) sera nécessaire. Les données extraites sont divisées en trois ensembles distincts : train, valid, test. Avant l'entraînement, deux pré-traitements clés sont appliqués : Augmentation des données (uniquement sur train) et Standardisation/ normalisation des

caractéristiques. Puis une analyse est menée pour identifier et conserver uniquement les caractéristiques les plus informatives. Les modèles sont entraînés sur train, puis évalués sur valid et test. Des métriques adaptées (précision, rappel, F1-score, etc.) permettent de mesurer leurs performances et de sélectionner la meilleure approche.

### 3.1.2. Extraire les caractéristiques

Pour chaque image, nous avons extrait des descripteurs manuels visant à capturer différents aspects visuels discriminants. Ces caractéristiques sont regroupées en plusieurs catégories :

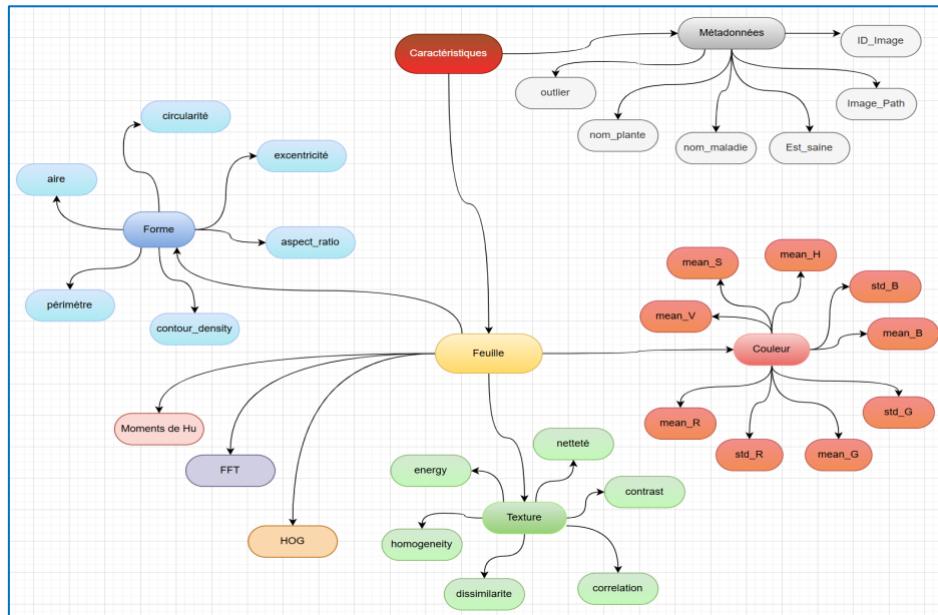


Figure 8 : Caractéristiques extraites des feuilles de plantes

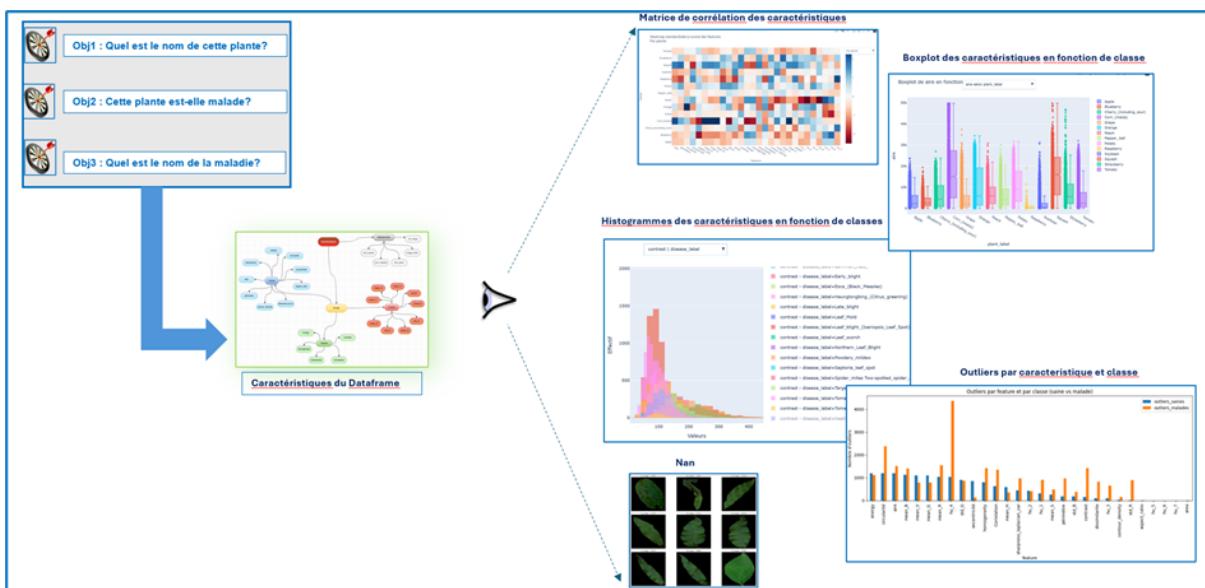
- **Caractéristiques morphologiques** : superficie (aire), périmètre, circularité, excentricité, rapport d'aspect (aspect ratio) et densité de contours. Ces indicateurs décrivent la forme globale des objets présents sur l'image.
- **Caractéristiques colorimétriques** : moyennes et écarts-types des canaux RGB (mean/std\_R, G, B), ainsi que les moyennes des composantes HSV (mean\_H, S, V), permettant de représenter les couleurs dominantes et leur variation.
- **Caractéristiques de texture** : netteté, contraste, energy, homogeneity, dissimilarity, correlation, calculées à partir de matrices de co-occurrence, décrivent les variations locales d'intensité dans l'image.
- **Descripteurs invariants** : les moments de Hu (hu\_1 à hu\_7) permettent de capturer la forme de manière invariante à la rotation, à la translation et au changement d'échelle.
- **Descripteurs fréquentiels** : les coefficients extraits de la transformée de Fourier (fft\_energy, fft\_entropy, low/high frequency power) communiquent une information sur la répartition spectrale des détails dans l'image.
- **Descripteurs de gradient** : les descripteurs HOG (moyenne, écart-type, entropie) résument les orientations dominantes des gradients, utiles pour capturer les structures visuelles.

Ces descripteurs sont concaténés pour former un vecteur unique par image, servant ensuite d'entrée aux algorithmes de classification. Les caractéristiques générées sont consignées dans un tableau (annexe 6.5), précisant pour chaque descripteur : sa source ou librairie d'origine, ainsi que sa fonction ou utilité dans le cadre de notre projet.

Certaines variables sont directement issues du traitement d'image (par exemple via OpenCV, NumPy, skimage.feature ou la transformée de Fourier), tandis que d'autres jouent un rôle de support (identifiant de la plante, label, cible, ou métadonnée de structure). Cette organisation facilite l'analyse, la traçabilité, ainsi que la future sélection des features les plus discriminantes pour la phase de classification.

### 3.1.3. Explorer les caractéristiques extraites

L'analyse exploratoire des données a permis d'examiner la qualité, la structure et les spécificités des images. Cette étape est cruciale pour identifier les défis potentiels et orienter les choix méthodologiques pour la modélisation. Le détail de l'analyse exploratoire se trouve en annexe 6.1.



#### Gestion des Données Manquantes et Outliers

- **Données manquantes** : 9 images présentaient des valeurs manquantes (NaN) et ont été supprimées. Ces images restent accessibles pour une analyse ultérieure si nécessaire.
- **Outliers** : La méthode IQR a révélé un taux élevé d'outliers (40,34% des données), principalement concentrés dans la classe "malade". Certaines caractéristiques comme la *circularité*, *fft\_entropy*, et *hu\_4* présentaient une surreprésentation marquée d'outliers pour les plantes malades.
- **Risques** : Ces outliers pourraient biaiser les modèles sensibles aux valeurs extrêmes ou refléter une hétérogénéité biologique légitime.
- **Recommandations** :
  - Tester l'impact de leur suppression ou transformation (ex. : winsorisation) sur les performances des modèles.
  - Privilégier des modèles robustes (ex. : Random Forest, SVM avec noyau RBF) ou des techniques de normalisation adaptées.

#### Distribution des Caractéristiques par Objectif

##### Objectif 1 : Identification de la Plante

- **Observations** : Les distributions des caractéristiques (ex. : mean\_G, mean\_H) varient significativement entre espèces, suggérant un fort potentiel discriminatif pour la classification. Certaines variables (ex. : canaux de couleur, moments de Hu) présentent une redondance, tandis que d'autres sont peu informatives pour certaines classes.
- **Risques** : Déséquilibre entre espèces risque de biaiser le modèle vers les classes majoritaires.
- **Recommandations** :
  - Sélection de variables pour réduire la redondance (ex. : Analyse de Corrélation, PCA).
  - Techniques de rééchantillonnage (ex. : SMOTE) ou pondération des classes pour atténuer le déséquilibre.

## Objectif 2 : Détection de l'État de Santé (Sain vs. Malade)

- **Observations** : Les variables morphologiques (aire, périmètre) et colorimétriques (mean\_R, mean\_G) montrent des différences nettes entre plantes saines et malades. Les plantes malades présentent souvent une aire réduite et des valeurs de couleur distinctes.
- **Recommandations** : Prioriser ces caractéristiques pour construire un modèle binaire robuste. Utiliser des visualisations interactives (ex. : boxplots) pour affiner la sélection des variables les plus discriminantes.

## Objectif 3 : Diagnostic Spécifique de la Maladie

- **Observations** : Forte disparité dans la représentation des maladies : certaines sont surreprésentées (ex. : Apple\_scab), tandis que d'autres ne comptent que quelques exemples. Des caractéristiques comme la contour\_density permettent de distinguer certaines maladies (ex. : Apple\_scab vs. Black\_rot).
- **Risques** : Déséquilibre extrême entre maladies, risquant de limiter la détection des pathologies rares.
- **Recommandations** :
  - Data Augmentation ciblée pour les maladies sous-représentées.
  - Modèles hiérarchiques : Classifier d'abord l'espèce, puis la maladie pour améliorer la précision.

## Relations entre Caractéristiques

- **Corrélations** : Les heatmaps ont révélé des groupes de variables fortement corrélées (ex. : canaux de couleur, moments de Hu), ainsi que des profils distincts entre espèces et maladies. Les plantes saines présentent des profils de caractéristiques plus homogènes, tandis que les plantes malades montrent des déviations marquées pour certaines variables.
- **Risques** : Redondance entre variables pourrait entraîner un surapprentissage. Hétérogénéité intra-classe complique la généralisation des modèles.
- **Recommandations** :
  - Réduction de dimensionnalité (PCA, t-SNE) pour visualiser et sélectionner les variables les plus pertinentes.
  - Ingénierie de caractéristiques pour combiner les variables redondantes en indices synthétiques (ex. : ratio de couleurs, indices de texture).

## Conclusions

- Les caractéristiques morphologiques et colorimétriques sont fortement discriminantes pour distinguer les plantes saines des malades.
- Le déséquilibre des classes (espèces/maladies) nécessite des stratégies de rééchantillonnage ou des modèles adaptés.

- La présence d'outliers et de redondances impose une préparation rigoureuse des données avant modélisation.

Les boxplots et heatmaps générés (disponibles en HTML) offrent une base solide pour affiner l'analyse et guider le choix des caractéristiques les plus pertinentes.

### 3.1.4. Ré-échantillonnage

Face au déséquilibre de classe, nous avons testé plusieurs techniques en fonction des modèles ML testés :

- RandomOverSampler vs SMOTE pour SVM-RBF : les meilleurs couples sont RandomOverSampler+StandardScaler et SMOTE+StandardScaler ( $f1\_macro\ CV \approx 0.8864$ ), alors que RobustScaler est en retrait ( $\sim 0.866$ ).

Nous avons créé un fichier csv comprenant le résultat de rééchantillonnage avant le split pour équilibrer les classes :

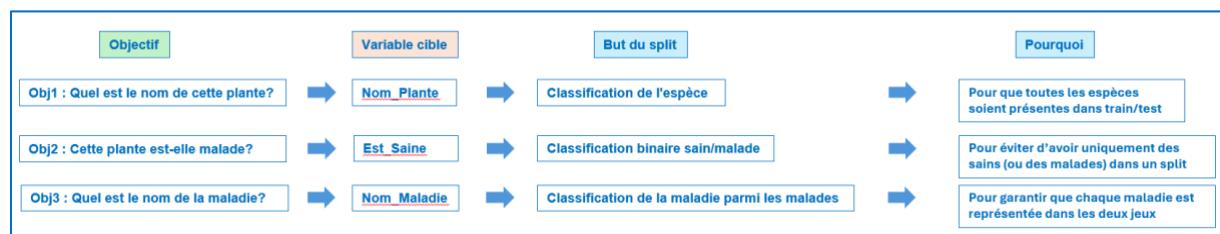
- oversampling ciblé des espèces/maladies minoritaires
- undersampling léger des classes sur-représentées

Ce fichier sera utilisé si les modèles explorés le requièrent.

### 3.1.5. Split train/test (et éventuellement validation)

Pour garantir la qualité de notre modèle, nous avons séparé le dataset en trois parties : un ensemble d'entraînement pour apprendre, un ensemble de validation pour régler et améliorer le modèle, et un ensemble de test pour évaluer ses performances réelles sur des données totalement nouvelles.

Le déséquilibre des classes constaté nous impose par ailleurs des découpages du jeu de données (train, val, test) stratifiés, c'est-à-dire en respectant la proportion des classes cibles pour chaque objectif :



### 3.1.6. Pré-traitements

#### 3.1.6.1. Data augmentation

Afin d'améliorer la robustesse et la généralisation de nos modèles nous avons réalisé de la data-augmentation, sur train uniquement, après découpage. Ceci afin de :

- Réduire le déséquilibre (même après rééchantillonnage certaines classes restent peu nombreuses).
- Enrichir la diversité des cas et forcer le modèle à travailler sur des variantes
- Limiter le sur-apprentissage en exposant le modèle à du bruit pour diminuer sa tendance à sur-apprendre

Nous avons appliqué une combinaison des méthodes suivantes :

- Flip (retournement horizontal/vertical)
- Rotation (ex :  $-30^\circ$ ,  $+30^\circ$ ...)
- Crop aléatoire (recadrage)
- Translation (décalage)
- Modification de la luminosité, contraste, bruit
- Zoom in/out
- Transformation couleur (changement de teinte, saturation)
- Gaussian blur, sharpness

Nous avons maintenant 91770 images

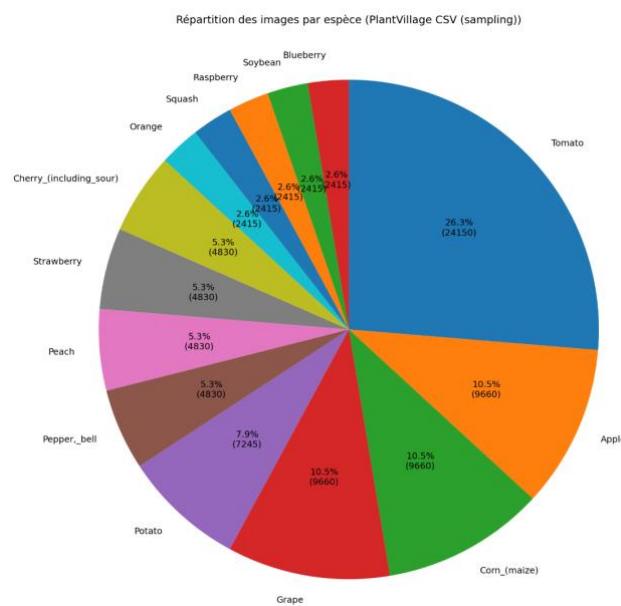


Figure 9 : Nouvelle répartition avec la data augmentation

### 3.1.6.2. Standardisation / normalisation des features

Lors de l'analyse exploratoire, nous avons observé un taux d'outliers élevé pour certaines variables (hu\_4, circularité, aire, mean\_R, mean\_B, etc.) ce qui implique que les méthodes "classiques" de scaling (standardisation, min-max) seront fortement influencées par ces valeurs extrêmes. Afin de garantir que nos modèles ne soient pas biaisés par des valeurs extrêmes ou des ordres de grandeurs différents nous avons :

- Appliqué un RobustScaler fit uniquement sur l'ensemble d'entraînement
- Transformé ensuite les ensembles de validation et de test avec ce même scaler préalablement ajusté.

L'analyse des distributions de certaines variables montrent que certaines variables avaient des ordres de grandeur très différents (ex: aire entre 0 et 50000, circularité entre 0 et 1).

### 3.1.7. Sélection des meilleures caractéristiques

Cette phase consiste à retenir les caractéristiques les plus pertinentes. On cherche à réduire la dimension du dataset, éliminer le bruit ou les variables non informatives, améliorer la performance et la vitesse des modèles. Nous avons comparé plusieurs familles de méthodes de sélection de variables (filtre, wrapper, intégré) pour mesurer l'importance des caractéristiques (voir liste des méthodes annexe 6.2), à l'aide d'un pipeline par sélecteur avec standardisation (RobustScaler), validation croisée stratifiée (StratifiedKFold) et un seul classifieur commun : RandomForestClassifier pour rendre les comparaisons cohérentes. Raisons du choix : robustesse au sur-apprentissage, gestion des classes déséquilibrées (class\_weight='balanced'), importance de variables intégrée et capacité à modéliser des relations non linéaires. Les métriques suivies incluent accuracy, F1 macro, précision et rappel. Les deux graphiques suivants présentent les résultats obtenus.

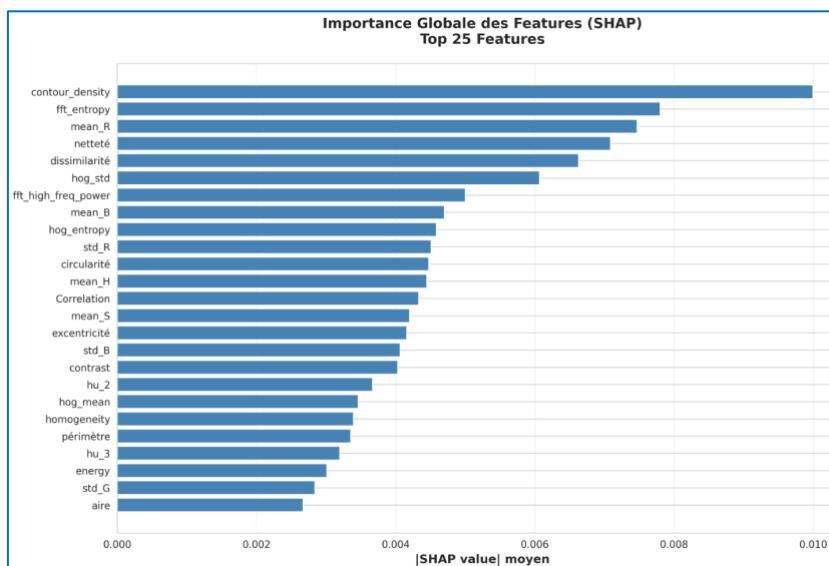


Figure 10 : Importance globale des variables (SHAP) — Top 25

La densité de contour (contour\_density) domine très nettement l'importance globale des features, avec une valeur SHAP moyenne de 0.010, soit environ 30% supérieure à la deuxième feature la plus importante. Les features de fréquence spectrale (fft\_entropy) et de couleur (mean\_R, mean\_B) occupent le trio de tête, confirmant que la forme des lésions et leurs caractéristiques colorimétriques sont les indicateurs les plus discriminants pour la classification des maladies.

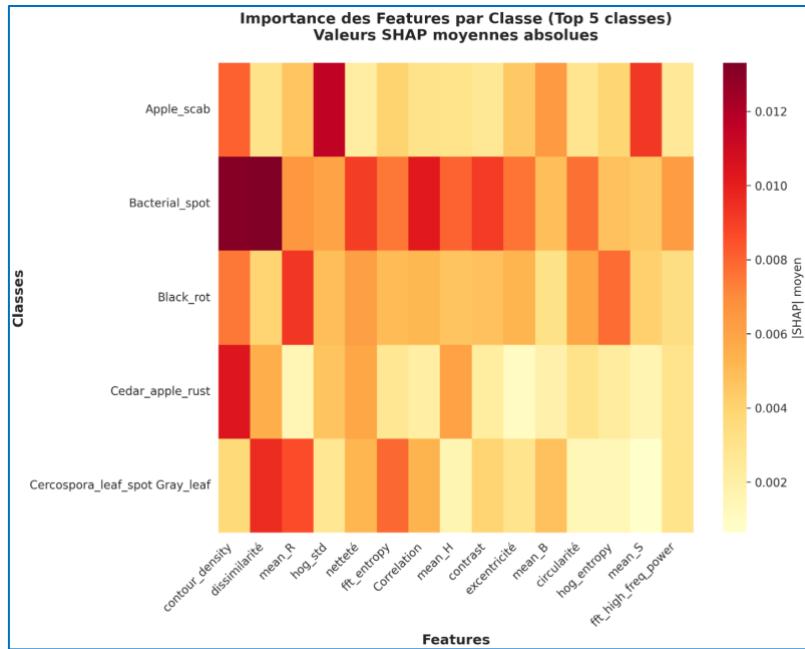


Figure 11: Importance des features par classe (SHAP) — Top-5 classes

L'analyse par classe révèle des signatures de features distinctes pour chaque maladie : par exemple, `hog_std` (texture) est extrêmement discriminant pour `Apple_scab` (valeur SHAP ~-0.013) mais beaucoup moins pour les autres maladies. À l'inverse, `contour_density` présente une importance élevée et relativement uniforme pour plusieurs maladies, indiquant qu'il s'agit d'une feature généraliste importante pour détecter les anomalies foliaires. Cette variabilité confirme que différentes maladies se manifestent par des combinaisons spécifiques de caractéristiques visuelles (forme, couleur, texture).

En synthèse, ces 2 graphiques nous apprennent :

- Qu'aucune catégorie unique ne suffit - il faut combiner : la géométrie (contour, circularité, excentricité), la couleur (mean\_R, mean\_B, std\_R), la texture (HOG descriptors) et la fréquence (FFT entropy et power)
- Que chaque classe de maladie s'appuie sur un sous-ensemble différent de features, justifiant l'utilisation d'un modèle complexe capable de capturer ces patterns spécifiques
- Que les 34 features extraites sont toutes pertinentes, aucune n'est totalement négligeable (même les moins importantes dans le top 25 contribuent).

### 3.1.8. Entrainement des modèles et évaluations

Nous avons réparti le travail en équipe avec un modèle par membre (SVM, XGBoost, Extra-Trees, Régression Logistique) pour la classification des plantes. Chaque membre a adopté sa propre méthodologie : optimisation poussée des hyperparamètres (exploration bayésienne, GridSearch), configurations variées, ou traitement simultané des trois objectifs (espèce, santé, maladie). Pour permettre une comparaison équitable, nous présentons ici uniquement les résultats de l'objectif 1 : identification de l'espèce, permettant d'évaluer transversalement les performances et la robustesse de chaque approche.

### Méthodologie commune :

- Dataset: plantvillage/segmented nettoyé
- Standardisation/normalisation des caractéristiques
- Split Train/Val/Test (stratifié)
- Validation croisée stratifiée (ex. : 5-fold) pour éviter le surapprentissage.

### Métriques choisies

Pour la classification binaire (plante saine vs malade) : précision, le rappel, et le F1-score. Ces métriques permettent de mesurer respectivement la capacité du modèle à éviter les faux positifs, à détecter correctement les cas positifs (malades), et à trouver un équilibre entre les deux, ce qui est crucial pour limiter les erreurs coûteuses (ex. : fausse détection de santé pour une plante malade).

Pour les problèmes multi-classes (ex. : identification de la plante, de la maladie), l'accuracy globale peut être trompeuse en raison des déséquilibres ; nous utiliserons donc la moyenne des F1-scores par classe (macro-F1) pour garantir une évaluation équitable de toutes les catégories, y compris les maladies rares.

Enfin, la matrice de confusion permettra d'identifier les confusions fréquentes entre classes. Ces métriques, combinées à une validation croisée stratifiée, assureront une évaluation rigoureuse et adaptée aux spécificités des données, notamment la présence d'outliers et de classes minoritaires.

### **Interprétation des Performances Globales**

Les résultats sont présentés dans le tableau comparatif synthétisant les performances pour l'**objectif 1 : Classification de la plante** :

Modèle	Accuracy	Précision (macro avg)	Rappel (macro avg)	F1-score (macro avg)
SVM(RBFVC)	0.9370	0.9271	0.9207	0.9237
XGBoosT	0.9038	0.9051	0.8654	0.8839
Régression Logistique	0.8615	0.8462	0.8214	0.8328
Extra-Trees	0.8310	0.8607	0.7405	0.7863

Tableau 2 : Performances des modèles ML sur l'ensemble de test

Le SVM-RBF démontre une excellente capacité de généralisation sur l'ensemble des 14 classes d'espèces. L'équilibre entre précision et rappel témoigne de sa robustesse face à la complexité et au déséquilibre du dataset. XGBoost occupe la deuxième position avec environ 4 points en-dessous du SVM-RBF, avec un rappel légèrement plus faible suggérant quelques difficultés à identifier certaines classes minoritaires. Enfin, Extra-Trees affiche les résultats les plus faibles, avec un rappel particulièrement faible indiquant des difficultés importantes à reconnaître correctement l'ensemble des espèces.

### **Interprétation des Performances par Classe**

Pour une analyse plus fine, les performances ont été détaillées par classe de plante comme détaillé par la figure suivante :

	precision	recall	f1-score	support
Apple	0.8777	0.8861	0.8819	632
Blueberry	0.9628	0.9500	0.9564	300
Cherry_(including_sour)	0.8898	0.8688	0.8792	381
Corn_(maize)	0.9730	0.9857	0.9793	769
Grape	0.9526	0.9643	0.9584	812
Orange	0.9596	0.9506	0.9548	1101
Peach	0.9371	0.9248	0.9369	532
Pepper_bell	0.8972	0.8300	0.8623	494
Potato	0.8897	0.8628	0.8760	430
Raspberry	0.8933	0.9054	0.8993	74
Soybean	0.9289	0.9371	0.9330	1018
Squash	0.9617	0.9591	0.9604	367
Strawberry	0.9103	0.9073	0.9088	313
Tomato	0.9455	0.9578	0.9516	3626
accuracy			0.9370	10849
macro avg	0.9271	0.9207	0.9237	10849
weighted avg	0.9368	0.9370	0.9368	10849

SVM-RBF

	precision	recall	f1-score	support
0	0.8185	0.7848	0.8013	632
1	0.9250	0.8633	0.8931	300
2	0.8963	0.8163	0.8544	381
3	0.9662	0.9662	0.9662	770
4	0.8942	0.9041	0.8991	813
5	0.9301	0.9310	0.9305	1101
6	0.9475	0.8816	0.9133	532
7	0.8558	0.7192	0.7816	495
8	0.8819	0.8163	0.8478	430
9	0.9683	0.8243	0.8905	74
10	0.9118	0.9342	0.9229	1018
11	0.9284	0.9183	0.9233	367
12	0.8480	0.8019	0.8243	313
13	0.8991	0.9548	0.9261	3629
accuracy			0.9038	10855
macro avg	0.9051	0.8654	0.8839	10855
weighted avg	0.9034	0.9038	0.9028	10855

XGBoost

	precision	recall	f1-score	support
Apple	0.7288	0.7057	0.7170	632
Blueberry	0.8845	0.8933	0.8889	300
Cherry_(including_sour)	0.7425	0.7113	0.7265	381
Corn_(maize)	0.9673	0.9623	0.9648	769
Grape	0.8613	0.8645	0.8629	812
Orange	0.8887	0.8783	0.8795	1101
Peach	0.8950	0.8496	0.8717	532
Pepper_bell	0.8025	0.6417	0.7132	494
Potato	0.8133	0.7698	0.7909	430
Raspberry	0.8529	0.7838	0.8169	74
Soybean	0.8335	0.8851	0.8585	1018
Squash	0.9036	0.8937	0.8986	367
Strawberry	0.8000	0.7412	0.7695	313
Tomato	0.8813	0.9192	0.8998	3626
accuracy			0.8615	10849
macro avg	0.8462	0.8214	0.8328	10849
weighted avg	0.8603	0.8615	0.8601	10849

Logistique régression

	precision	recall	f1-score	support
Apple	0.8631	0.5585	0.6782	632
Blueberry	0.9176	0.5200	0.6638	300
Cherry_(including_sour)	0.8399	0.7297	0.7809	381
Corn_(maize)	0.9331	0.9610	0.9468	769
Grape	0.8154	0.8103	0.8128	812
Orange	0.8773	0.8765	0.8769	1101
Peach	0.9362	0.7726	0.8465	532
Pepper_bell	0.7657	0.4433	0.5615	494
Potato	0.8148	0.6651	0.7324	430
Raspberry	0.9388	0.6216	0.7488	74
Soybean	0.8373	0.9047	0.8697	1018
Squash	0.9081	0.8883	0.8981	367
Strawberry	0.8189	0.6645	0.7337	313
Tomato	0.7835	0.9512	0.8592	3626
accuracy			0.8310	10849
macro avg	0.8607	0.7405	0.7863	10849
weighted avg	0.8356	0.8310	0.8238	10849

Extra-Trees

Figure 12 : Performances des modèles ML par classe

L'analyse par classe révèle des disparités significatives dans les capacités de classification des différents modèles. Les espèces les mieux reconnues sont Blueberry, Tomato et Grape, qui bénéficient d'un support important (respectivement 300, 3626 et 812 échantillons) et probablement de caractéristiques visuelles distinctives. À l'inverse, certaines espèces posent des défis considérables à tous les modèles, notamment Pepper\_bell, Potato et Strawberry, suggérant une ambiguïté intrinsèque dans leurs caractéristiques morphologiques ou une variabilité intra-classe élevée. Le déséquilibre des classes (support variant de 74 à 3626) impacte particulièrement les modèles les moins robustes : Extra-Trees et Régression Logistique peinent sur les classes minoritaires comme Raspberry (74 échantillons) avec des f1-scores respectifs de 0.748 et 0.817, alors que SVM-RBF maintient 0.890. Les métriques macro avg (moyenne non pondérée) versus weighted avg (moyenne pondérée) confirment que SVM-RBF gère mieux l'équilibre entre toutes les classes, tandis que les autres modèles sont davantage influencés par les classes majoritaires.

On trouvera dans l'annexe 6.3 un tableau de comparaison des caractéristiques des modèles.

## Discussion des Résultats

Identifier automatiquement les espèces de plantes est réalisable avec une accuracy supérieure à 93% (SVM-RBF). La supériorité du SVM avec noyau RBF s'explique par sa capacité à capturer des relations non-linéaires complexes, essentielles pour distinguer les subtilités morphologiques entre espèces. L'écart de 4 points avec XGBoost confirme que les caractéristiques extraïtes bénéficient davantage d'une transformation par noyau que d'approches ensemblistes.

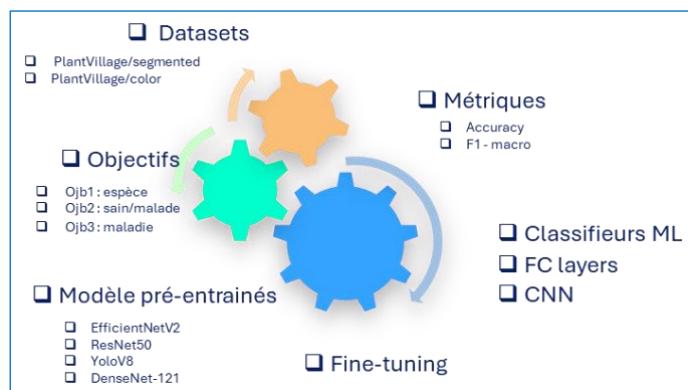
Les difficultés sur certaines classes (Pepper\_bell, Potato, Strawberry) suggèrent des pistes d'amélioration : augmentation ciblée des données. Le déséquilibre des classes reste un défi malgré l'optimisation des hyperparamètres et aux poids de classes ajustés de SVM-RBF.

Bien que XGBoost et Extra-Trees pourraient bénéficier d'une optimisation plus poussée (exploration bayésienne, GridSearch exhaustif), le SVM-RBF constitue le choix optimal pour l'identification d'espèces végétales, offrant un excellent compromis entre performance globale et équité entre classes.

## 3.2. Deep Learning

Dans le cadre de notre formation, nous avons procédé en deux étapes :

**Etape1** : chaque membre de l'équipe a expérimenté les techniques de Deep Learning pour se confronter aux diverses problématiques en jouant sur les leviers suivants :



**Etape 2** : Nous avons pratiqué une démarche structurée présentée dans ce qui suit :

### 3.2.1. Méthodologie



Afin de comprendre le fonctionnement du Deep Learning et ses défis, nous avons explorés 9 architectures. Pour effectuer une évaluation comparative, nous avons restreint le nombre d'architectures pour couvrir notre scénario avec 3 cas. Enfin, la sélection et recommandation est une projection « s'il fallait faire un déploiement ».

### 3.2.2. Définition des critères de sélection

Pour rappel, le scénario se compose de 3 cas :

- Cas 1 — Identification d'espèce : Seule l'espèce doit être identifiée sans diagnostic de maladie (ex: Botaniste identifie juste l'espèce)
- Cas 2 — Diagnostic ciblé : L'espèce de la plante est connue, seule l'identification de la maladie est requise (ex: Agriculteur connaît sa plante, veut le diagnostic)

- Cas 3 — Diagnostic complet : L'espèce et la maladie sont inconnues, nécessitant une identification complète (ex: Application tout public sans connaissance préalable)

Les architectures explorées retenues seront évaluées sur les critères suivants :

Classe de critère	Critère	Justification
Critères métier	Précision (Macro-F1)	Mesure la capacité du modèle à bien prédire toutes les classes, y compris les rares. Essentiel pour un diagnostic fiable.
	Précision (Accuracy)	Pourcentage de prédictions correctes. Indicateur simple mais peut être trompeur si déséquilibre de classes. À compléter par le Macro-F1.
	Généralisation (écart val/test)	Différence entre performances validation et test. Écart faible (<2%) = modèle robuste qui généralise bien. Écart élevé (>5%) = surapprentissage (overfitting).
	Couverture opérationnelle	Capacité à répondre aux 3 cas d'usage métier : (1) diagnostic maladie si plante connue, (2) identification espèce, (3) diagnostic complet. Détermine la polyvalence du modèle déployé.
Critères techniques	Coût d'inférence (FLOPs)	FLOPs (relatif) ≈ nombre de passes du backbone par image. Plus de FLOPs = plus de calculs = batterie consommée. Critique pour déploiement mobile/edge (smartphones agriculteurs, drones).
	Coût d'inférence (latence)	Temps réel pour obtenir une prédiction (en millisecondes). Impacte l'expérience utilisateur : <100 ms = fluide, >500 ms = frustrant.
	Coût d'entraînement (temps)	Important pour itérer rapidement lors des expérimentations et améliorer le modèle.
	Coût d'entraînement (GPU)	Ressources matérielles nécessaires (VRAM, type de GPU). Détermine le budget cloud ou la nécessité d'un GPU local.
	Complexité (paramètres)	Nombre de poids à stocker. Impact sur la taille du fichier modèle (déploiement) et le temps de chargement.
Critères pédagogiques	Complexité (maintenabilité)	Facilité à comprendre, modifier et debugger le code. Important pour collaboration et évolution future.
	Concepts explorés	Variété des techniques deep learning testées
Critères additionnels	Apprentissages acquis	Compétences concrètes grâce au projet
	Interprétabilité	Capacité à expliquer les prédictions du modèle (ex: Grad-CAM pour visualiser zones d'attention).
	Besoins en données	Quantité d'images annotées minimale pour entraîner efficacement.

Tableau 3 : Critères de sélection des architectures

### 3.2.3. Exploration d'architectures DL

Nous avons choisi d'utiliser le transfert d'apprentissage car les modèles sont déjà entraînés sur des millions d'images pour détecter des motifs génériques (contours, textures, formes). De plus, c'est un gain de temps et de ressources. Cela nous évite de partir de zéro. Le tableau suivant présente les caractéristiques des modèles pré-entraînés retenus :

Caractéristique	EfficientNetV2-S	ResNet50	YOLOv8n-cls*	DenseNet-121
Année	2021	2015	2023	2017
Auteurs/Org	Google Brain	Microsoft Research	Ultralytics	Cornell/Facebook
Paramètres (M)	21.5	25.6	2.7	8.0
Taille modèle (MB)	~86	~102	~11	~32
GFLOPs (224×224)	8.4	4.1	4.2	2.9
GFLOPs (256×256)	~10.8	~5.3	~5.4	~3.7
Taille vecteur sortie	1280	2048	1024	1024
Top-1 Acc ImageNet	83.9%	76.1%	69.0%	74.4%
Top-5 Acc ImageNet	96.7%	93.0%	88.3%	92.0%
Latence CPU (ms)	60-80	40-50	25-35	30-40
Latence GPU (ms)	5-8	3-5	2-4	3-5
Taille entrée	384×384 (optim.)	224×224	224×224	224×224
Profondeur (layers)	~150	50	~100	121

Tableau 4 : Comparatif des caractéristiques des modèles pré-entraînés choisis

En analysant les caractéristiques des modèles pré-entraînés, **EfficientNetV2S**, pré-entraîné avec ImageNet, nous a semblé un bon compromis.

## Protocole expérimental

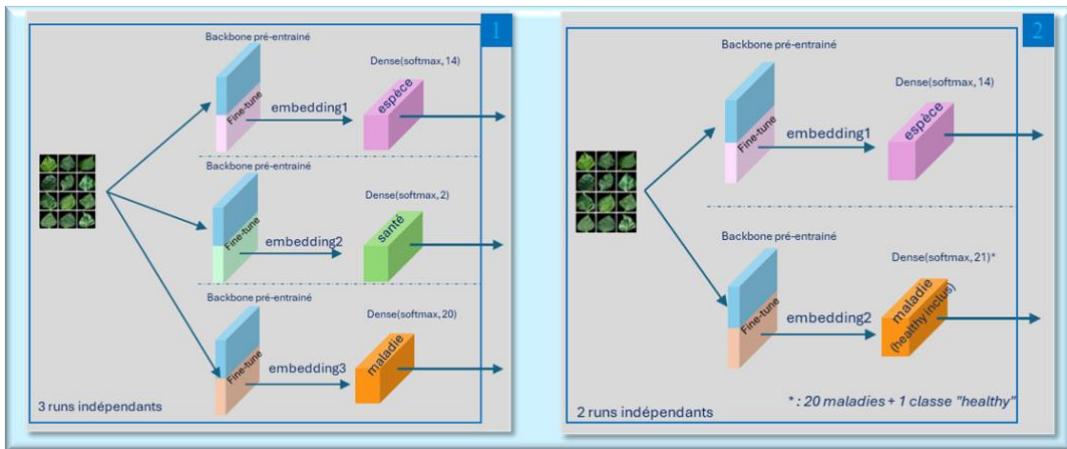
- Le même dataset PlantVillage/color
- Splits identiques pour tous les modèles (fichier pv\_color\_splits.json)
- Hyperparamètres fixés : learning rate, batch size, augmentation (pour comparaison équitable)
- Le même Backbone pré-entraîné EfficientNetV2S (ImageNet)
- Métriques trackées : Loss, Accuracy, Macro-F1 (par tête si multi-objectif), matrice de confusion
- Seuil de Sur-apprentissage : Nous considérons comme équilibré  $|Accuracy - Macro-F1| \leq 0,002$ . Entre 0,002 et 0,005: acceptable mais à surveiller.  $>0,005$ : déséquilibre notable à investiguer.

## Entrainements réalisés

- 1 run par architecture (préparation, stage 1, stage 2, évaluation finale)
- Logs sont sauvegardés (courbes, history.csv)
- Checkpoint du meilleur modèle : (val\_macro\_f1)

Les architectures explorées sont issues de nos nombreuses discussions et de nos lectures. Pour les présenter, elles sont réparties en 2 groupes : les architectures dont le backbone pré-entraîné est dédié à chaque objectif (tête), et les architectures dont le backbone pré-entraîné est partagé entre plusieurs objectifs. Pour éviter les répétitions : les classes considérées sont : 14 espèces (species), 2 états de santé(healthy), 20 maladies(disease). Parcours du dataset, extraction des features par le backbone, constitution des splits train/val/test. Les courbes des métriques trackées sont en annexe 6.4.

## Backbone pré-entraîné dédié à chaque objectif :



### Architecture 1 :

**Architecture spécialisée** : Trois modèles CNN indépendants, chacun dédié à une seule tâche (species, health, disease). Chaque modèle comprend un backbone pré-entraîné et une tête de classification Dense adaptée au nombre de classes.

**Workflow** : Chaque modèle s'entraîne en 2 phases sur le même dataset d'entrée : (1) backbone gelé avec entraînement de la tête uniquement; (2) fine-tuning des dernières couches du backbone pour adapter les features ImageNet aux spécificités du dataset.

**Avantages** : Simplicité (1 tâche = 1 modèle), absence de conflits entre tâches (pas de compromis dans l'optimisation), performances maximales par tâche (spécialisation totale), interprétabilité facilitée (1 objectif clair par modèle).

**Limites** : Triplication des ressources (3 backbones à stocker et maintenir), inférences multiples pour cas d'usage complexes, absence de synergie inter-tâches (pas de transfert d'apprentissage entre les 3 têtes), temps d'entraînement cumulé plus long (3 runs).

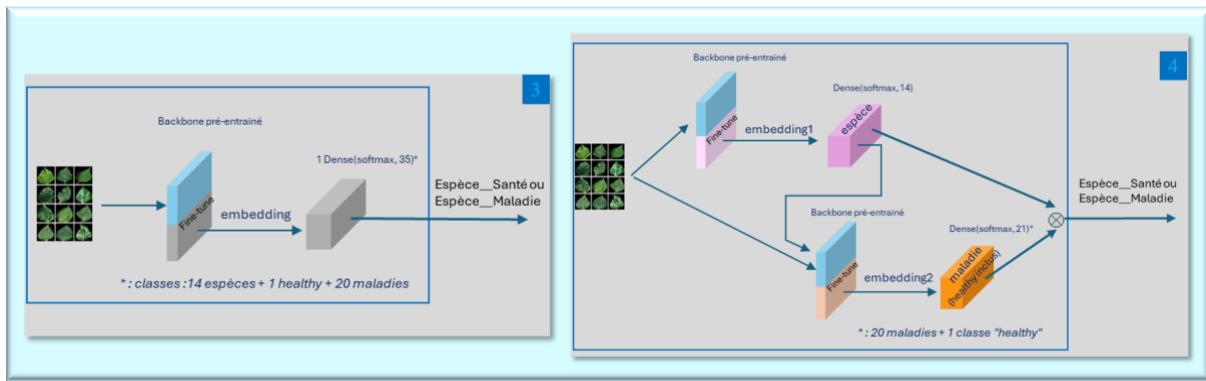
### Architecture 2 :

**Architecture** : Deux modèles CNN indépendants, l'un pour l'espèce, l'autre pour l'état sanitaire complet : la classe "healthy" est intégrée comme une maladie spéciale.

**Workflow** : Deux runs mono-tâche. Le modèle species s'entraîne sur toutes les images (saines + malades). Le modèle disease\_extended s'entraîne également sur toutes les images.

**Avantages** : Simplicité (2 têtes), uniformité (deux softmax multi-classe), diagnostic complet en 2 inférences (species + disease\_extended), "healthy" est un état sanitaire comme les maladies.

**Limites** : Déséquilibre accru (classe "healthy" majoritaire), perte de la métrique binaire explicite healthy/diseased, interprétation plus ambiguë des prédictions mixtes (ex: 40% healthy, 35% early\_blight).



### Architecture 3 :

**Architecture unifiée :** Un modèle CNN pré-entraîné + 1 tête Dense softmax (35 classes). Étiquette combinée : chaque image est étiquetée par un couple “Espèce\_Etat” (Tomato\_healthy, Apple\_scab...).

**Workflow :** Phase 1: backbone gelé, entraînement de la tête uniquement. Phase 2: fine-tuning partiel des dernières couches du backbone. Les labels sont pré-combinés en 35 classes.

**Avantages :** Un seul modèle, une seule inférence : plus simple à déployer et à utiliser. Synergie entre tâches : l'apprentissage capte directement les co-dépendances espèce↔maladie/santé.

**Limites :** Moins de spécialisation par tâche. Les classes rares peuvent être sous-apprises. Peu flexible : impossible de gérer des paires inédites (nouvelle espèce/maladie) sans réentraîner les 35 classes. Interprétabilité : plus dur d'isoler l'erreur (vient-elle de l'identification d'espèce ou de maladie ?).

### Architecture 4 :

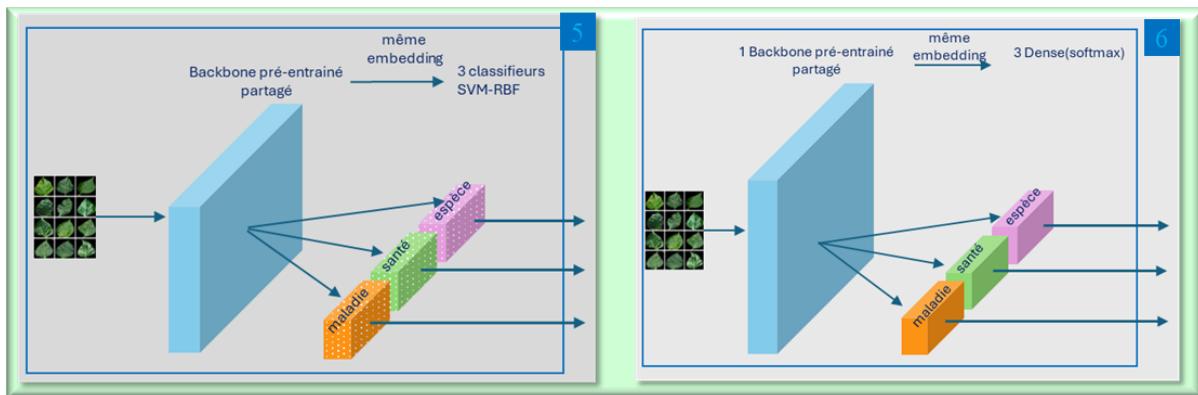
**Architecture en cascade :** Deux modèles CNN pré-entraînés sont chaînés. Un classificateur d'espèce extrait un embedding d'image et prédit l'espèce. Un classificateur de maladie global (21 classes, dont “healthy”) reçoit l'image + l'espèce True et applique une attention spatiale pour se focaliser sur les zones pertinentes, puis prédit la maladie. À l'inférence, la maladie est conditionnée par l'espèce prédite. La sortie finale se lit comme une étiquette composée “Espèce\_Santé/Maladie”.

**Workflow :** Phase 1 : backbone gelé, entraînement de la tête. Phase 2 : fine-tuning partiel du backbone. Entraînement du modèle maladie en 2 phases, avec la même logique (tête puis fine-tuning), mais en lui fournissant l'espèce (True) en entrée pour stabiliser l'apprentissage. Évaluation en CASCADE - maladie avec espèce prédite (performance réelle de bout en bout).

**Avantages :** La prédiction d'espèce guide la maladie, réduisant les confusions entre espèces. L'attention spatiale sur la branche maladie aide à capturer les indices visuels pertinents. Modularité : possibilité d'améliorer séparément espèce ou maladie sans tout réentraîner.

**Limites :** Une espèce mal prédite dégrade la maladie. Le modèle maladie voit l'espèce (True) à l'entraînement mais la prédite en production. Latence accrue avec passes réseau successives. En cas d'espèce erronée, une maladie impossible peut être proposée.

## Backbone pré-entraîné partagé entre plusieurs objectifs :



### Architecture 5

**Architecture “CNN + SVM” :** Un backbone CNN pré-entraîné (gelé) transforme chaque image en vecteur d’embeddings (features). Des classificateurs SVM (espèce, santé, maladie) sont entraînés sur ces embeddings.

**Workflow :** Sauvegarde des vecteurs + labels. Puis chargement des embeddings, entraînement de trois têtes SVM: Espèce: multi-classe, Santé: binaire (healthy vs diseased), Maladie : soit global (multi-classe, malades uniquement), soit par espèce (un SVM par espèce, malades uniquement).

**Avantages :** Entraînement très rapide des SVM; itérations légères (on réutilise les embeddings). Simplicité opérationnelle : séparation claire “features gelées” / “classificateurs”; facile de remplacer le backbone ou de réentraîner seulement les SVM.

**Limites :** Les features restent génériques : pas d’adaptation conjointe aux tâches du dataset. Cohérence multi-tâches limitée.

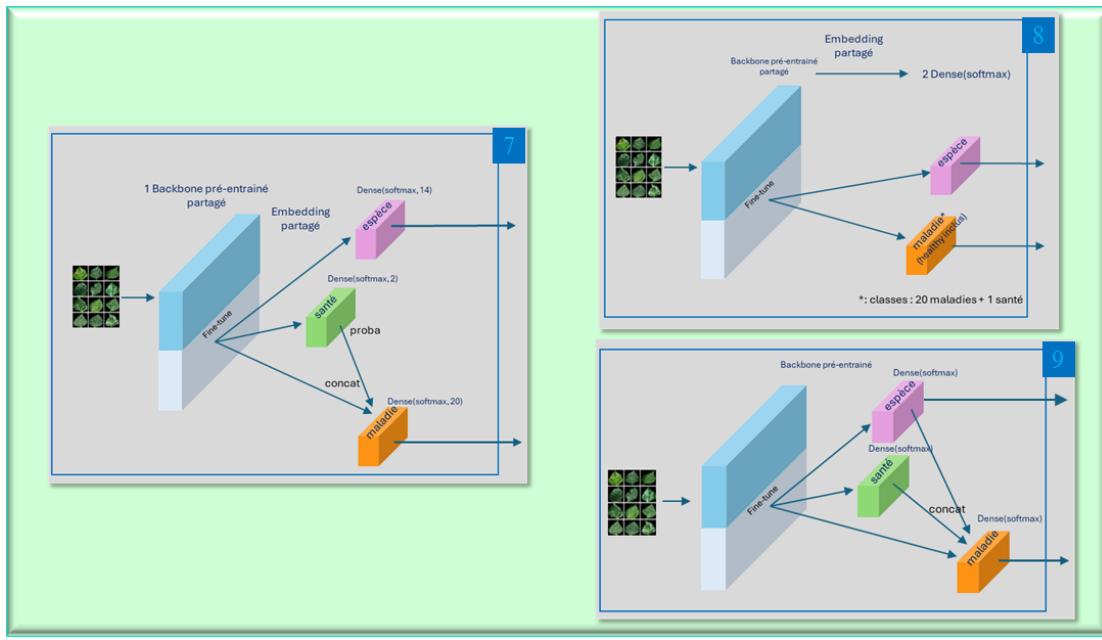
### Architecture 6

**Architecture multi-tâche unifiée :** Un seul backbone CNN pré-entraîné, partagé, produit un embedding commun, puis trois têtes de classification parallèles: Espèce, Santé, Maladie. La tête “maladie” est optimisée sur les images malades, afin d’éviter de perturber l’apprentissage par des exemples “healthy”.

**Workflow :** Une seule phase “têtes seules” avec backbone gelé (pertes pondérées par tête).

**Avantages :** Les trois tâches se renforcent (l’espèce et la santé aident la maladie). Un seul backbone à entraîner ; une seule inférence pour obtenir espèce, santé, maladie. Contrôle des compromis : pondérations de pertes par tête pour équilibrer objectifs (espèce/santé/maladie).

**Limites :** Conflits d’optimisation : objectifs parfois concurrents ; sensibilité aux pondérations des pertes. Malgré la tête dédiée, les maladies peu représentées restent difficiles. On n’active pas le fine-tuning, les features ImageNet peuvent rester trop génériques, la Loss du cas fine-tuning était catastrophique. Couplage des tâches : une mauvaise modélisation de l’espèce/santé peut impacter indirectement la maladie (via le partage d’embedding).



## Architecture 7

**Architecture multi-tâche à 2 têtes:** Un backbone CNN pré-entraîné partagé produit un embedding commun. Tête espèce: multi-classe. Tête maladie: multi-classe hors “healthy”, activée uniquement pour les échantillons malades. Santé (healthy/diseased) n'est pas une sortie directe: un signal santé auxiliaire interne (probabilité “malade”) est appris et injecté comme feature dans la tête maladie pour l'aider à se focaliser sur les cas réellement malades.

**Workflow :** Phase 1: entraînement des têtes avec backbone gelé (pondérations de pertes, l'échantillon tagué “healthy” n'entraîne pas la tête maladie). Phase 2: fine-tuning partiel des couches hautes du backbone pour adapter les features au domaine.

**Avantages :** Une seule passe backbone pour deux tâches; coût d'inférence réduit. L'injection de la probabilité “malade” et le masquage de perte évitent que les “healthy” perturbent la tête maladie. Synergie utile: l'embedding partagé bénéficie des signaux espèce et santé auxiliaire, améliorant la discrimination des maladies. Equilibre des objectifs via pondérations des pertes;

**Limites :** Pas de sortie santé explicite: pas de score/label “healthy vs diseased” livrable tel quel (signal interne non calibré pour un usage direct). Dépendance au signal santé: si le signal auxiliaire est biaisé, la tête maladie peut sur- ou sous-activer certaines classes. Conflits d'optimisation: partage d'un même backbone pour deux objectifs; sensibilité aux pondérations et au fine-tuning. Classes rares: malgré le masquage des “healthy”, les maladies peu représentées restent difficiles.

## Architecture 8

**Architecture multi-tâche simplifiée (2 têtes) :** Un seul backbone CNN pré-entraîné, partagé, et deux têtes parallèles: Espèce, Disease qui inclut explicitement healthy. Pas de tête “santé” dédiée, pas de masquage d'échantillons: toutes les images (saines et malades) entraînent les deux têtes.

**Workflow** : Préparation: splits stratifiés par dossier de classe, pipeline tf.data avec augmentations légères et prétraitement compatible EfficientNetV2S. Entraînement: Phase 1: entraînement des têtes avec backbone gelé (pondérations de pertes, label smoothing). Phase 2: fine-tuning partiel du haut du backbone (option gradient clipping) pour adapter les features au domaine. Évaluation: accuracy et macro-F1 pour l'espèce et pour disease\_all (les 21 classes), matrices de confusion, courbes, rapport Markdown. Inférence: une seule passe réseau → deux sorties simultanées: Espèce et Healthy/Maladie\_k.

**Avantages** : Simplicité: pas de tête santé, pas de règles/mask; supervision uniforme sur toutes les images. Efficience: un seul backbone et une seule inférence pour obtenir espèce + santé/maladie. Cohérence de décision: healthy fait partie du même espace que les maladies → seuils et calibration unifiés au sein d'une softmax à 21 classes. Maintenance légère: pipeline standardisé (figures, rapports, checkpoints) et fine-tuning optionnel.

**Limites** : Déséquilibre "healthy": la classe healthy peut dominer et biaiser la tête disease\_all, au détriment des maladies rares (macro-F1 crucial). Pas de conditionnement par espèce: la tête maladie n'est pas contrainte par l'espèce → risque de confusions inter-espèces. Seuils globaux: frontières de décision communes à toutes les espèces; calibration potentiellement sous-optimale pour des distributions très différentes selon l'espèce. Shortcut possible: le modèle peut exploiter des corrélations de fond (espèce, contexte) plutôt que des lésions fines si les données sont biaisées.

## Architecture 9

**Architecture conditionnée (Species + Health → Disease)**: Un backbone CNN pré-entraîné unique produit un embedding partagé. Tête espèce: multi-classe. Tête maladie: multi-classe (hors "healthy"), conditionnée par deux signaux additionnels: le vecteur de probabilités d'espèce. la probabilité interne d'être malade (tête santé auxiliaire non exposée). Les échantillons "healthy" n'entraînent pas la tête maladie (masque de perte).

**Workflow** : Phase 1: apprentissage des têtes avec backbone gelé, pondérations de pertes,. Phase 2: fine-tuning partiel des couches hautes. La tête maladie est optimisée uniquement sur les images malades (healthy masqués).

**Avantages** : Conditionnement explicite: la maladie est guidée par l'info d'espèce et un indicateur de santé, ce qui réduit les confusions inter-espèces et focalise sur les cas réellement malades. Synergie multi-tâches: l'embedding partagé + signaux auxiliaires apportent un contexte fort au classifieur maladie. Efficience: un seul backbone à entraîner/déployer; une seule inférence pour obtenir espèce et maladie. Contrôle des compromis: pondérations de pertes par tête; fine-tuning optionnel selon budget/performance.

**Limites** : Propagation d'erreurs: une erreur d'espèce ou un biais du signal santé peut entraîner une mauvaise prédiction de maladie. Raccourcis/biais: le modèle peut sur-utiliser les a priori espèce/santé au détriment d'indices visuels fins si les données sont déséquilibrées. Pas de sortie santé livrable: la santé est un signal interne; si un score "healthy vs diseased" est requis, il faut une tête/mesure dédiée. Calibration sur "healthy": la tête maladie n'est pas entraînée sur les sains; ses sorties peuvent être peu informatives pour des images réellement "healthy" si utilisées seules.

## Synthèse des performances des 9 architectures

Le surapprentissage a été vérifié sur toutes les architectures. Le tableau suivant présente les performances obtenues :

Arch	Nom	Espèce		Maladie	
		Macro-F1	Accuracy	Macro-F1	Accuracy
1	Mono-tâche 3 modèles séparés	0.9990	0.9991	0.9897	0.9939
2	Mono-tâche 2 modèles séparés	0.9990	0.9991	0.9924	0.9964
3	Mono-tâche 1 modèle	0.9990	0.9990	0.9931	0.9968
4	Cascade 2 modèles	0.9988	0.9989	0.9933	0.9971
5	Embeddings + SVM	0.9976	0.9978	0.9758	0.9850
6	Multi-tâche (3 têtes)	0.9900	1.0000	0.95	0.96
7	Multi-tâche (2 têtes+ concat santé)	0.9985	0.9988	0.9978	0.9927
8	Multi-tâche (2 têtes)	0.9980	0.9983	0.9864	0.9917
9	Multi-tâche (2 têtes + concat spèce et santé )	0.9988	0.9989	0.9922	0.9951

Tableau 5 : Synthèse des performances des 9 architectures

Les meilleures performances sont obtenues par les architectures 1, 3, 7 et 9, avec des Macro-F1 et des accuracies proches de 0.99 pour l'espèce et des accuracies  $\geq 0.99$  pour la maladie. L'architecture 3 combine une mise en production simple avec d'excellents résultats (espèce et maladie), très compétitive face aux approches multi-têtes.

La complexité de la cascade de l'architecture 4 n'est pas justifiée face aux architectures 3/7/9 déjà performantes et plus simples pour la maintenance. L'architecture 6 est nettement en retrait sur la maladie par rapport aux alternatives. Cette faiblesse sur la tête maladie rend l'usage opérationnel moins fiable alors que d'autres architectures atteignent  $\approx 0.99$ . L'architecture 8 n'apporte pas de gain mesurable vis-à-vis des architectures 7 et 9 et reste plus basse sur la maladie ( $\approx 0.986$  de Macro-F1 vs  $\geq 0.989$ ). En conséquence, pour la suite, les architectures 4,6 et 8 sont exclues.

## Synthèse des coûts des 9 architectures

Archi	Description	FLOPs (relatif)	Latence (indicative)	Entrainement: temps	Entrainement: GPU	Paramètres (M)	Maintenabilité
1	Mono-tâche 3 modèles séparés	1x (Cas1/2) • 3x (Cas3)	faible (Cas1/2) • élevé (Cas3, 3 passes)	élevé (3 modèles)	modéré	136 (*3 en Cas3)	faible (3 pipelines)
2	Multi-tâche 3 têtes (partagé)	1x	faible	moyen	modéré	137	bonne
3	Mono-tâche 1 tête (35 classes)	1x	faible	moyen	modéré	136	très bonne (pipeline simple)
4	Cascade 2 modèles	2x	élevée (2 inférences)	élevé (2 modèles)	modéré à élevé	2 backbones (B0+B2)	moyenne (2 étages)
5	Embeddings + SVM	1x	faible	faible à moyen (SVM rapide; dépend si backbone gelé)	modéré	136 (+SVM négl.)	moyenne (mix DL/ML)
6	Multi-tâche 3 têtes (sans fine-tuning)	1x	faible	faible à moyen (pas de FT)	modéré à faible	137	bonne
7	Multi-tâche 2 têtes	1x	faible	moyen	modéré	137	bonne
8	Multi-tâche 2 têtes (health=classe)	1x	faible	moyen	modéré	137	bonne
9	Arch9 species>health>disease	1x	faible	moyen	modéré	137	moyenne (logique hiérarchique)

Les architectures 2/7/8 offrent le meilleur compromis (1x FLOPs, latence faible, entraînement moyen) et une bonne maintenabilité, ce qui en fait des candidats naturels pour le déploiement temps réel. L'architecture 4 (deux passes B0→B2+attention) augmente nettement la latence et le coût d'inférence avec un risque d'erreur propagée. L'architecture 3 (mono-tête 35 classes) maximise la maintenabilité avec un pipeline unique et un coût modéré. L'architecture 5 (embeddings+SVM) allie inférence rapide et entraînement léger au prix d'un stack mixte DL/ML, l'architecture 6 (sans FT) réduit fortement temps/GPU mais peut plafonner en performance, et l'architecture 9 ajoute une complexité hiérarchique qui pèse sur la maintenabilité.

Décision :

Nous excluons **l'architecture 4** car sa cascade en deux passes augmente nettement la latence et la complexité sans gain tangible face aux architectures 3/7/9 déjà proches de 0.99, avec en plus un risque de propagation d'erreurs. Nous écartons **l'architecture 6** car elle est nettement en retrait sur la maladie, ce qui dégrade la fiabilité opérationnelle face à des alternatives qui atteignent  $\approx 0.99$  et  $\geq 0.989$  de Macro-F1. Enfin, **l'architecture 8** n'apporte pas de bénéfice mesurable par rapport aux 7/9 et reste plus basse sur la maladie ( $\approx 0.986$  de Macro-F1 vs  $\geq 0.989$ ), ne justifiant pas sa complexité supplémentaire.

### 3.2.4. Évaluation comparative

L'évaluation est présentée en fonction des 3 cas de notre scénario global.

**Cas 1 — Identification d'espèce :** Seule l'espèce doit être identifiée sans diagnostic de maladie (ex: Botaniste identifie juste l'espèce)

Pour les 6 architectures restantes, comparons Accuracy et Macro F1-Score ainsi que la corrélation Accuracy vs F1 pour valider la cohérence :

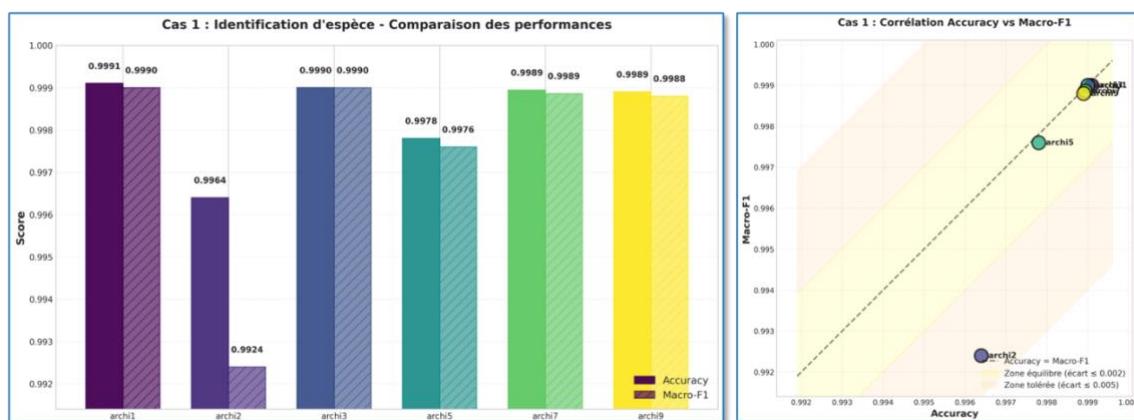


Figure 13 : comparaison et corrélation Accuracy et Macro F1-Score par architecture - cas 1

Toutes les architectures atteignent une précision quasi parfaite pour l'identification d'espèce, avec des écarts très faibles entre elles.

Les écarts Accuracy – Macro-F1 sont quasi nuls pour archi1/3/5/7/9 ( $\leq 0.0002$ ), ce qui implique une performance très homogène entre espèces et un risque faible de biais de classe. Archi2 présente un écart plus élevé (~0.004), signe que quelques espèces (souvent minoritaires) tirent le Macro-F1 vers le bas.

Voici le classement final des 6 architectures pour l'identification de l'espèce, basé sur le Macro F1 plus robuste que l'Accuracy.

Rang	Archi	Nom	Accuracy	Macro-F1
#1	archi1	Mono-tâche 3 modèles séparés	0.9991	0.9990
#2	archi3	Mono-tâche 1 tête (35 classes)	0.9990	0.9990
#3	archi7	Multi-tâche 2 têtes	0.9989	0.9989
#4	archi9	Arch9 species→health→disease	0.9989	0.9988
#5	archi5	CNN + SVM (embeddings)	0.9978	0.9976
#6	archi2	Mono-tâche 2 têtes (disease→species)	0.9964	0.9924

Tableau 6 : classement final des 6 architectures pour l'identification de l'espèce

**Cas 2 — Diagnostique ciblé :** L'espèce de la plante est connue, seule l'identification de la maladie est requise (ex: Agriculteur connaît sa plante, veut le diagnostic)

Pour les 6 architectures restantes, comparons Accuracy et Macro F1-Score ainsi que la corrélation Accuracy vs F1 pour valider la cohérence :

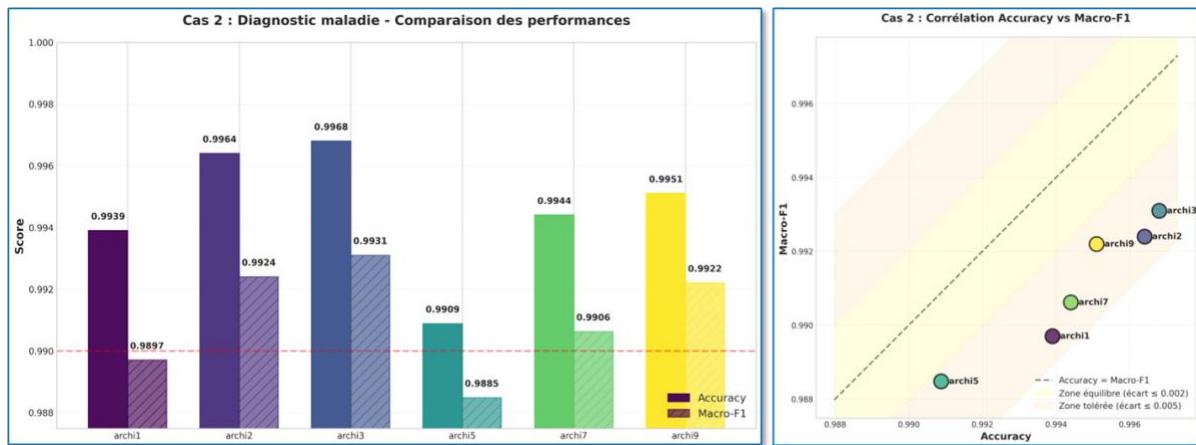


Figure 14 : comparaison et corrélation Accuracy et Macro F1-Score par architecture - cas 2

Les architectures archi3 et archi2 affichent les meilleures précisions et Macro-F1, avec archi9 très proche derrière. Archi5 est en retrait (Macro-F1 la plus basse), tandis que archi1 et archi7 restent élevés mais légèrement sous le trio de tête.

Tous les modèles sont en dehors de la bande d'équilibre mais dans la zone tolérée (écart ~0.002–0.004), ce qui traduit une Accuracy systématiquement supérieure au Macro-F1: certaines maladies sont un peu moins bien reconnues (risque de disparités par classe). Cela peut se traduire par des risques accrus de faux négatifs ou faux positifs sur des maladies rares.

Voici le classement final des 6 architectures pour l'identification de l'espèce. Le classement est basé sur le Macro F1-Score, qui est plus robuste que l'Accuracy.

Rang	Archi	Nom	Accuracy	Macro-F1
#1	archi3	Mono-tâche 1 tête (35 classes)	0.9968	0.9931
#2	archi2	Mono-tâche 2 têtes (disease→species)	0.9964	0.9924
#3	archi9	Arch9 species→health→disease	0.9951	0.9922
#4	archi7	Multi-tâche 2 têtes	0.9944	0.9906
#5	archi1	Mono-tâche 3 modèles séparés	0.9939	0.9897
#6	archi5	CNN + SVM (embeddings)	0.9909	0.9885

Tableau 7 : classement final des 6 architectures pour l'identification de la maladie

**Cas 3 — Diagnostic complet :** L'espèce et la maladie sont inconnues, nécessitant une identification complète (ex : Application grand public sans connaissance préalable)

Pour les 6 architectures restantes, comparons Accuracy et Macro F1-Score ainsi que la corrélation Accuracy vs F1 pour valider la cohérence :

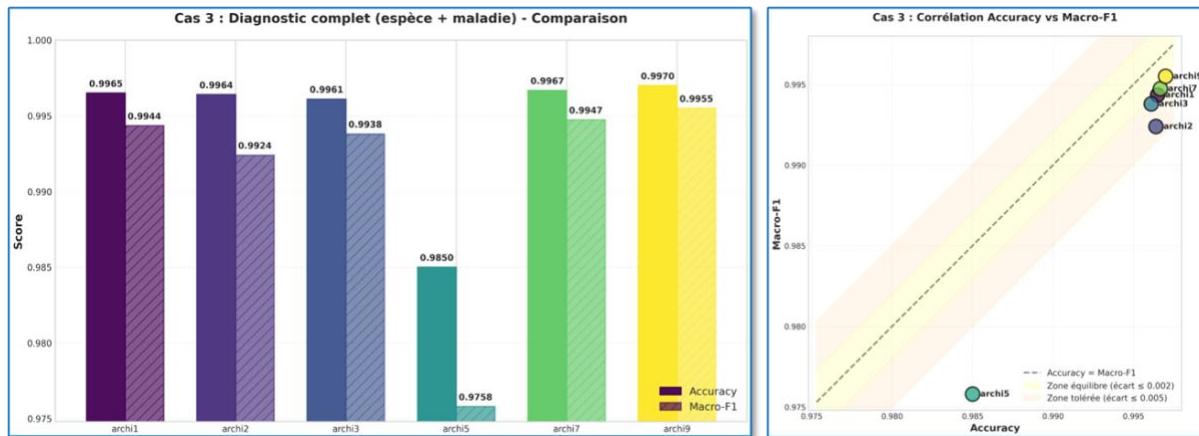


Figure 15 : comparaison et corrélation Accuracy et Macro F1-Score par architecture - cas 3

Pour un diagnostic bout-à-bout, archi9 offre le meilleur compromis (Accuracy≈0.997, Macro-F1≈0.9955), avec archi7 puis archi1 très proches; archi5 est nettement en retrait et à éviter pour un usage critique.

Les écarts Acc-F1 sont contenus pour archi9/7/1/3 (<0.0025) mais plus marqués pour archi2 (~0.004) et surtout archi5 (~0.009), impliquant un risque d'hétérogénéité par classes qui requiert calibration/pondération des pertes, augmentation ciblée et monitoring par maladie si déployés. Les architectures archi7 et archi9 restent dans l'équilibre ( $\leq 0.002$ ), indiquant un comportement cohérent sur les deux tâches ; archi1 est limite ( $\approx 0.0021$ ) et archi2 en toléré (~0.004). Ces modèles (surtout archi9, puis archi7) sont de bons candidats "prêts à déployer" pour un flux bout-à-bout avec moins de surprises par classe. L'archi2 nécessite un monitoring par classe ; archi5 (écart ~0.009) n'est pas recommandé sans amélioration substantielle.

Voici le classement final des 6 architectures pour l'identification de l'espèce. Le classement est basé sur le Macro F1-Score, qui est plus robuste que l'Accuracy.

Rang	Archi	Nom	Accuracy	Macro-F1
#1	archi9	Arch9 species→health→disease	0.9970	0.9955
#2	archi7	Multi-tâche 2 têtes	0.9967	0.9947
#3	archi1	Mono-tâche 3 modèles séparés	0.9965	0.9944
#4	archi3	Mono-tâche 1 tête (35 classes)	0.9961	0.9938
#5	archi2	Mono-tâche 2 têtes (disease→species)	0.9964	0.9924
#6	archi5	CNN + SVM (embeddings)	0.9850	0.9758

Tableau 8 : classement final des 6 architectures pour l'identification complète

### 3.2.5. Sélection et recommandations

La sélection et les recommandations se feront sur deux contextes de déploiement : Production standard - Applications professionnelles et Applications mobiles / Edge computing.

Compte tenu des classements précédents, nous ne retenons que les architectures : archi1, archi3, archi7, archi9.

La synthèse des résultats de tous les critères est présentée ci-dessous avec le jeu de couleurs suivant :

- Vert clair :  $F1 \geq 0.995$  (excellent).
- Jaune clair :  $0.990 \leq F1 < 0.995$  (très bon).
- Orange clair :  $0.970 \leq F1 < 0.990$  (correct).

Métrique	Mono 3 modèles (Arch 1)	Mono 1 modèle (Arch 3)	Multi 2 têtes (Arch 7)	Multi 2 têtes (Arch 9)
<b>F1 moyen</b>	99.44%	99.53%	99.47%	99.55%
<b>F1 species</b>	99.90%	99.57%	99.89%	99.88%
<b>F1 disease</b>	98.97%	99.38%	99.06%	99.22%
<b>Accuracy moyen</b>	99.65%	99.73%	99.67%	99.70%
<b>Écart Acc-F1</b>	0.21%	0.20%	0.19%	0.15%
<b>Paramètres</b>	408M	137M	137M	137M
<b>Coût relatif</b>	3.0x	1.0x	1.0x	1.0x
<b>Nb modèles</b>	3	1	1	1
<b>Inférences</b>	3x	1x	2x	3x

Tableau 9 : Synthèse des critères évalués pour les architectures retenues

Le tableau suivant nous présente la synthèse des performances obtenues par cas :

Rang Global	Architecture	Nom	Cas 1 (Species)	Cas 2 (Disease)	Cas 3 (Complet)	F1 Moyen
#1	Arch 9	Arch9 species→health→disease	99.88%	99.22%	99.55%	99.55%
#2	Arch 3	Mono 35 classes	99.90%	99.31%	99.38%	99.53%
#3	Arch 7	Multi 2 têtes (seeds)	99.89%	99.06%	99.47%	99.47%
#4	Arch 1	Mono 3 têtes	99.90%	98.97%	99.44%	99.44%

Tableau 10 : synthèse des performances obtenues par cas

Pour un déploiement en Production standard – Applications professionnelles, nous sélectionnons l'architecture 9 (species→health→disease). Elle est classée #1 avec le meilleur F1 moyen (99.55%) et des scores équilibrés sur les trois cas, tout en présentant l'écart Accuracy–F1 le plus faible ( $\approx 0.15\%$ ), signe d'une bonne calibration et d'une robustesse opérationnelle. Avec 137M de paramètres (coût relatif 1.0x) et un seul modèle à maintenir, elle reste fiable malgré 3 inférences, grâce à la stabilité de son schéma multi-tâches. Exemple de profil d'utilisation recommandé : Applications web/serveur où un temps de réponse < 500 ms est acceptable, Systèmes de diagnostic professionnels pour botanistes, Déploiement cloud avec GPU, Budget d'hébergement : 1 modèle

Pour un déploiement sur Applications mobiles / Edge computing, nous retenons l'architecture 3 (Mono 35 classes). Classée #2, elle atteint un F1 moyen de 99,53% — quasi identique à l'archi9 — avec un pipeline minimal (1 modèle, 1 inférence) qui réduit latence et complexité. Le tableau de synthèse indique un coût relatif de 1,0x (137M params) et un écart Accuracy–F1

contenu ( $\approx 0,20\%$ ), offrant le meilleur compromis précision/complexité/latence pour un environnement contraint. La maintenance et l'intégration sont simplifiées grâce à la mono-tête. Exemple de profil d'utilisation recommandé : Applications mobiles (iOS/Android) avec contraintes de batterie, Systèmes embarqués (Raspberry Pi, Jetson Nano), Déploiement edge avec connectivité intermittente, Budget: minimal (1 seul modèle)

### 3.2.6. Limites et perspectives

Nous avons identifié 4 limitations clés :

1. Validation terrain absente : Test en conditions agricoles réelles / Validation par experts botanistes / Pilote terrain avec agriculteurs / Feedback utilisateurs finaux. En conséquence, les performances présentées (99.67%) constituent une borne supérieure en conditions contrôlées.
2. Biais du dataset PlantVillage : Fond uniforme (blanc/vert uni)/ Éclairage contrôlé (studio)/ Feuilles isolées, détournées/ Angles standardisés. Conséquence, une chute de performance - 5% à -15% en conditions réelles
3. Validation statistique limitée : 1 seul run par architecture / Seed aléatoire unique / Pas de test statistique (Test de significativité ( $p\_value < 0,05$ )). Conséquence : les résultats exacts peuvent varier ( $\pm 0.1\text{--}0.2\%$ )
4. Classes maladies difficiles identifiées : Faux négatifs : Maladie rare non détectée → propagation. Faux positifs : Sur-traitement inutile (coût + écologie)

En termes de perspectives, voici quelques actions présentées par ordre de priorité :

1. Analyser l'Interprétabilité avec Grad-CAM
2. Effectuer entre 3 et 5 Runs multiples + statistiques : si nous avons le temps, continuer de vérifier sur les architectures retenues leur robustesse
3. Analyser les classes difficiles
4. Augmenter le nombre de classes pour appréhender l'impact dans nos architectures retenues

## 4. Conclusion

La première phase du projet a reposé sur une approche méthodique, depuis l'analyse exploratoire des données brutes jusqu'à la sélection des caractéristiques les plus pertinentes. Le choix du dataset PlantVillage a permis une immersion efficace dans les outils de Machine Learning, tout en maîtrisant la complexité du sujet. Grâce à l'ingénierie de caractéristiques et à une analyse approfondie, nous avons enrichi notre compréhension des données, identifié leurs spécificités et leurs contraintes, et posé des bases solides pour la suite du projet.

Pour la phase de modélisation, influencée par les travaux de la publication « A Systematic Review of Deep Learning Techniques for Plant Diseases », nous avons exploré deux axes principaux : les modèles classiques de Machine Learning (SVM, XGBoost, Extra-Trees, Régression Logistique) et les modèles de Deep Learning.

Parmi les modèles classiques, le SVM-RBF s'est imposé comme le plus performant pour l'identification des 14 espèces, offrant le meilleur compromis précision/rappel et une robustesse face aux déséquilibres de classes. XGBoost et Extra-Trees ont également montré des résultats prometteurs, bien qu'un réglage plus poussé des hyperparamètres aurait pu optimiser leurs performances. Les analyses par classe ont révélé des disparités : d'excellents résultats sur les classes Blueberry, Tomato et Grape, mais des défis persistants pour des classes plus ambiguës (Pepper\_bell, Potato, Strawberry), ouvrant la voie à des améliorations ciblées (rééquilibrage, augmentation de données).

Pour appréhender la diversité des architectures et comprendre leur fonctionnement, nous avons d'abord défini et évalué 9 architectures distinctes. Cette phase initiale nous a permis de nous confronter aux défis propres au Deep Learning et d'identifier les forces et limites de chaque approche. À l'issue de cette première évaluation, 3 architectures ont été exclues en raison de leurs performances ou de leur inadéquation avec nos objectifs. Les 6 architectures restantes ont ensuite été testées de manière approfondie sur 3 cas d'usage opérationnels, représentant des scénarios variés. Bien que le déploiement ne fasse pas partie des exigences du projet, cette analyse nous a conduit à sélectionner 2 architectures parmi les plus performantes (archi9 et archi3), en vue d'un éventuel déploiement en production. Ces architectures ont été jugées les plus adaptées à des contextes variés, qu'il s'agisse d'applications professionnelles ou de solutions mobile/Edge computing.

Cette phase du projet a également mis en lumière des contraintes opérationnelles : la disposition des ressources nécessaires à l'exécution des modèles de Deep Learning et l'adaptation du code pour exploiter pleinement les GPU, l'arrivée tardive du cours sur MLFlow qui aurait facilité la gestion des nombreuses itérations.

Malgré des résultats très performants, nous sommes conscients des limites de notre travail. Il nous reste un point essentiel à réaliser : travailler l'interprétabilité de nos modèles avec des techniques telles que Grad-CAM.

## 5. Bibliographie

Publications :

- **[2024] - Trends in Machine and Deep Learning Techniques for Plant Disease Identification: A Systematic Review** - Diana-Carmen Rodríguez-Lira , Diana-Margarita Córdova-Esparza , José M. Álvarez-Alvarado Juan Terven , Julio-Alejandro Romero-González, JuvenalRodríguez-Reséndiz
- **[2024] - A systematic review of machine learning and deep learning approaches in plant species detection** - Deepti Barhate , Sunil Pathak, Bhupesh Kumar Singh , Amit Jain , Ashutosh Kumar Dubey
- **[2024] - A systematic review of deep learning techniques for plant diseases** - Ishak Pacal · Ismail Kunduracioglu · Mehmet Hakki Alma · Muhammet Deveci · Seifedine Kadry · Jan Nedoma · Vlastimil Slany · Radek Martinek
- **[2024] - ASystematic Literature Review of Machine Learning and Deep Learning Approaches for Spectral Image Classification in Agricultural Applications Using**

**Aerial Photography** – Usman Khan, MuhammadKhalidKhan, MuhammadAyubLatif, MuhammadNaveed, MuhammadMansoorAlam, SalmanA.Khan, MazlihamMohdSu'ud

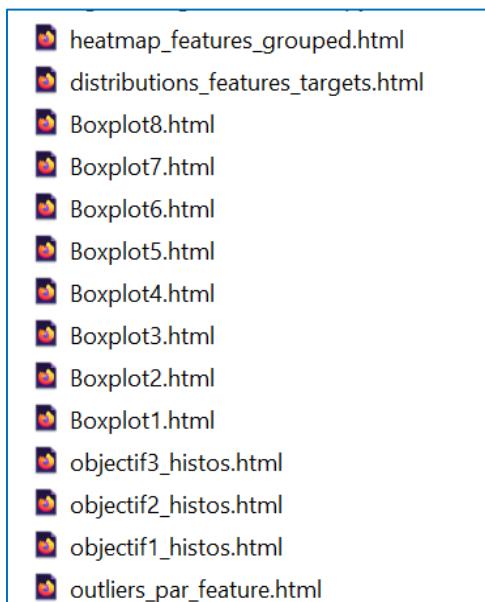
Livres :

- [2022] - **The StatQuest Illustrated Guide To Machine Learning** – Josh Starmer -
- [2017] – **Machine Learning avec Scikit-Learn** – Aurélien Géron - Dunod

## 6. Annexes

### 6.1. Analyse exploratoire des caractéristiques extraites

L'analyse exploratoire est supportée par des graphiques (histogrammes, boxplot, heatmap...). Ils sont générés avec plotly, avec des menus déroulants. Ils sont enregistrés dans des fichiers HTML pour explorer plus précisément les caractéristiques :



#### Gestion des données manquantes

Nous avons détecté 9 images avec NaN. Dans un premier temps, nous les avons supprimées. On pourra toujours les trouver si besoin.

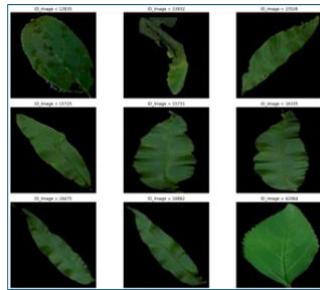


Figure : données manquantes

Identification des outliers : La détection des outliers est faite via la méthode IQR pour chaque variable. Nombre de lignes avec au moins un outlier: 21900.

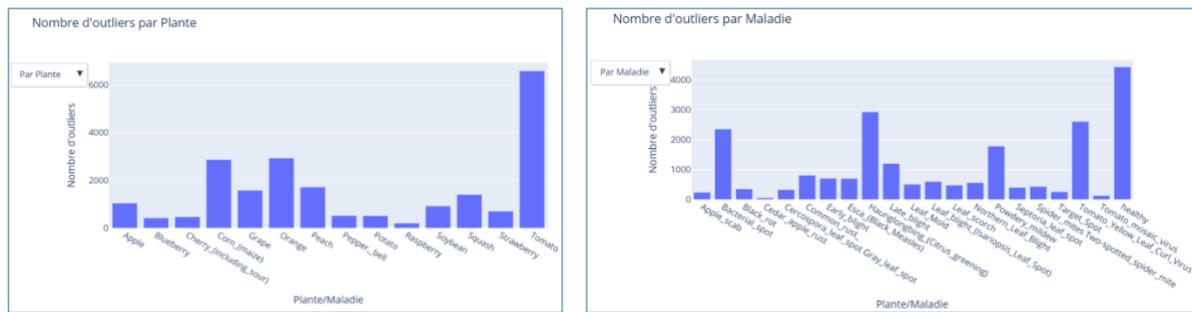


Figure : Identification des outliers par plante – identification des outliers par maladie

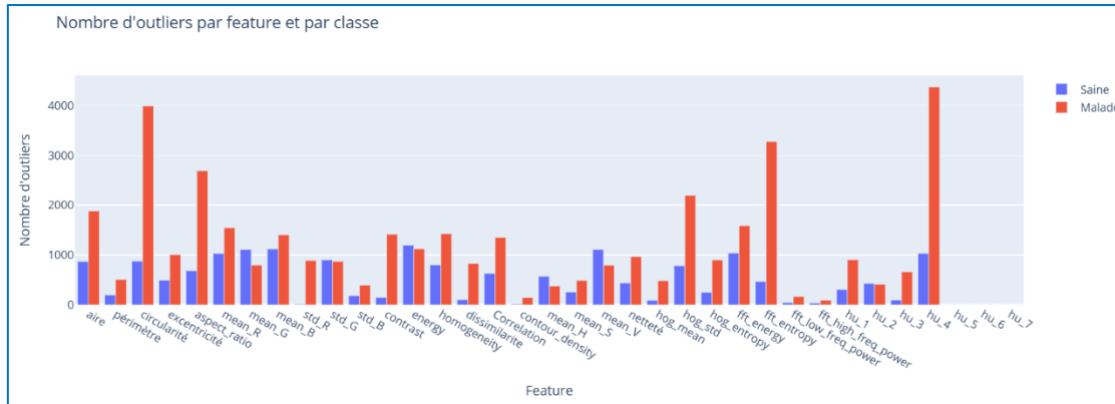


Figure : Identification des outliers par feature et par classe

### Observations :

Le taux d'outliers est élevé : 40,34%. On constate que, pour la majorité des variables, la classe "malade" affiche significativement plus d'outliers que la classe "saine". Circularité, fft\_entropy et hu\_4, se distinguent avec un nombre élevé d'outliers chez les malades.

### Risques :

La surreprésentation d'outliers chez les malades confirme l'hétérogénéité importante au sein de cette classe, mais cela peut aussi révéler des problèmes potentiels de qualité de données ou de mesures extrêmes spécifiques à certains cas qui risquent de biaiser les analyses

statistiques et les modèles prédictifs, en faussant la détection des vraies différences entre classes. Enfin, la présence d'un nombre très faible ou nul d'outliers sur certaines features pourrait signaler des variables peu discriminantes ou mal calibrées. Les features avec beaucoup d'outliers risquent de perturber certains algorithmes sensibles aux valeurs extrêmes. Supprimer les outliers, sans discernement, risque d'enlever des cas légitimes et importants. Possible présence de valeurs aberrantes parmi les outliers détectés.

### Conclusions :

Ces graphiques mettent en évidence l'importance de traiter et d'analyser spécifiquement les outliers. Avant toute modélisation, il sera essentiel d'investiguer l'origine de ces valeurs extrêmes, de décider de leur prise en compte (suppression, correction, transformation) ou utilisation de modèles robustes aux outliers. Analyse approfondie des outliers : Inspecter visuellement et statistiquement les valeurs extrêmes pour distinguer les vraies valeurs aberrantes (bruit, erreur) des observations atypiques mais valides. Tester l'impact du traitement des outliers sur la performance des modèles (avec/sans, transformation...). Prioriser les features discriminantes : Les variables qui montrent beaucoup d'outliers chez les malades pourraient être de bons marqueurs pour la classification santé/maladie.

Dans le chapitre sur les Représentations visuelles, les boxplots sont également des représentations pertinentes pour explorer la nature des outliers.

### Distribution des caractéristiques

Un jeu d'histogrammes interactifs permet l'exploration de la distribution des caractéristiques suivant l'objectif auquel le modèle devra répondre.

#### **Objectif1 : Quelle est cette plante ?**

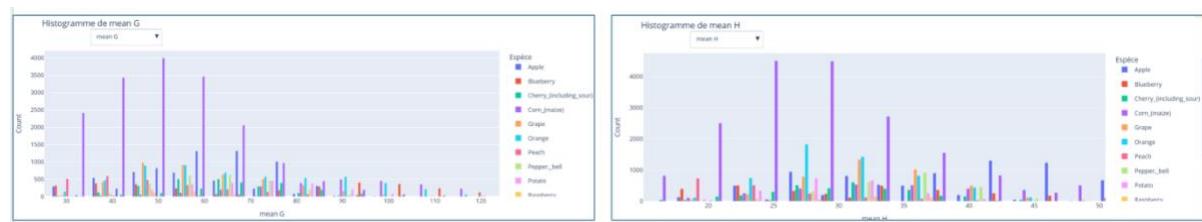


Figure : Distribution des caractéristiques pour objectif 1 "Quelle est cette plante?"

### Observations :

Variabilité importante entre espèces : Les distributions de certaines features (aire, couleur, texture...) varient fortement d'une plante à l'autre, ce qui suggère que plusieurs caractéristiques sont potentiellement très discriminantes pour l'identification de la plante.

Features très redondantes ou corrélées : Certaines familles de variables (ex : canaux couleurs, moments de Hu) présentent des corrélations élevées entre elles, ce qui peut indiquer une certaine redondance à traiter lors de la sélection des variables.

Caractéristiques quasi constantes ou peu informatives pour certaines classes : Certaines variables varient peu selon l'espèce, ce qui les rend peu utiles pour la classification de la plante (par exemple, une texture ou une couleur presque identique dans toutes les classes).

## Risques :

Un fort déséquilibre entre espèces peut biaiser le modèle, qui aura tendance à privilégier les classes majoritaires lors de la prédiction. Les espèces sous-représentées risquent d'être mal reconnues, augmentant le taux d'erreur pour ces classes. Les performances globales du modèle peuvent être trompeuses et masquer des faiblesses sur les minorités. Ce déséquilibre complique la généralisation et la robustesse de la solution sur l'ensemble des plantes du jeu de données.

## Conclusions :

On observe que la variabilité de certaines caractéristiques entre espèces offre de bons leviers pour la classification. Toutefois, la redondance, la présence d'outliers et des variables peu informatives devront être prises en compte pour optimiser la robustesse et la performance du modèle pour l'objectif "identifier la plante".

## **Objectif2 : Cette plante est-elle saine ?**

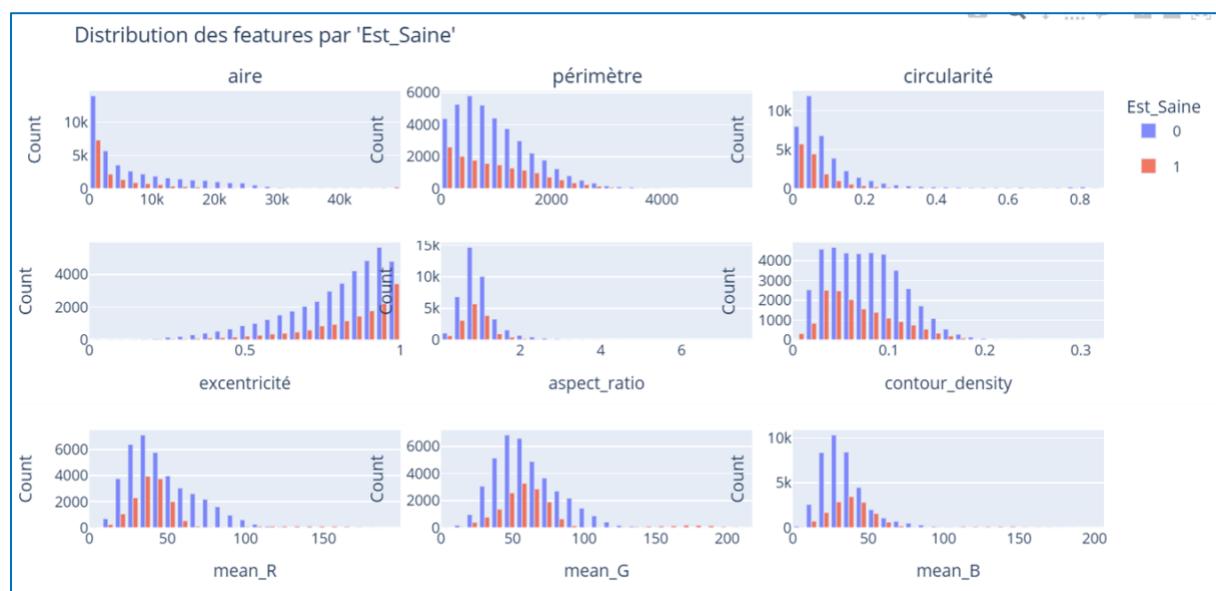


Figure : Distribution des caractéristiques pour objectif 2 (1/3)

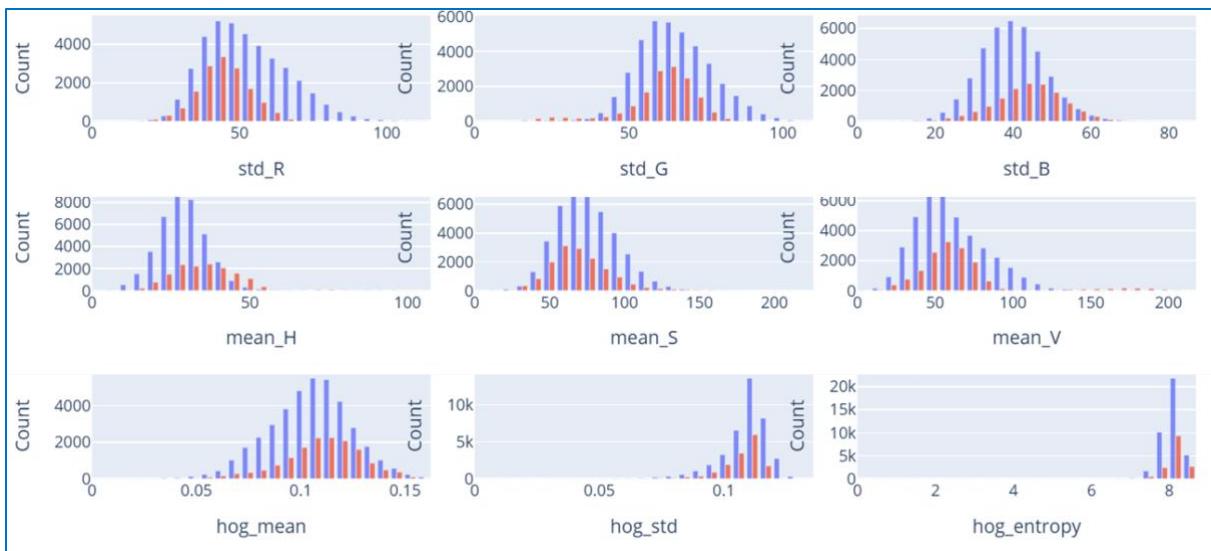


Figure : Distribution des caractéristiques pour objectif 2 (2/3)

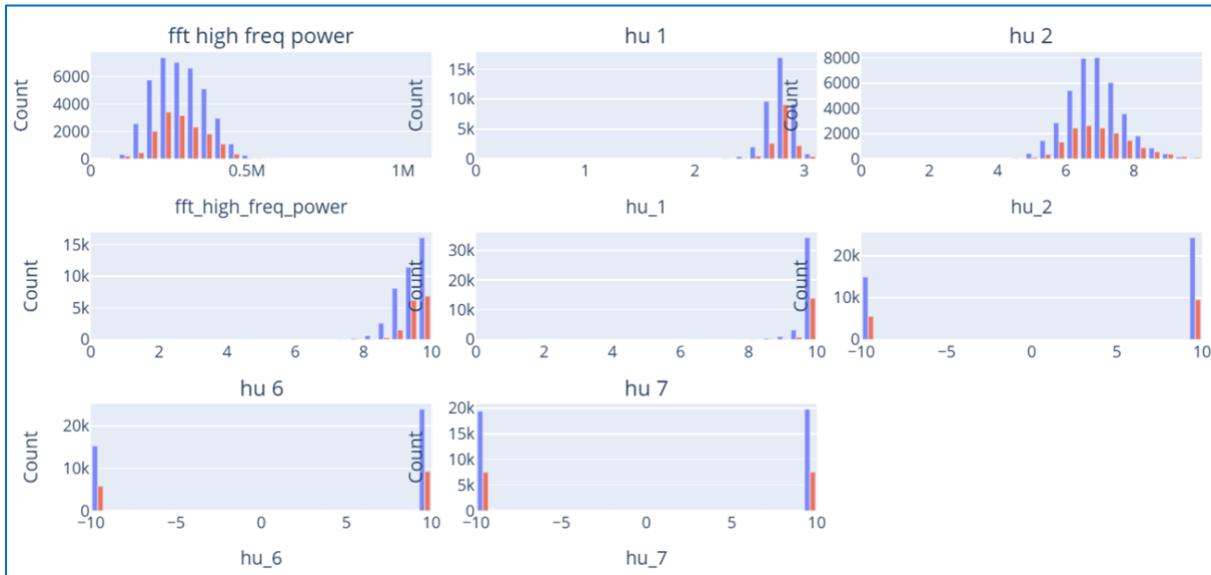


Figure : Distribution des caractéristiques pour objectif 2 (3/3)

#### Observations :

Les variables morphologiques (aire, périmètre, circularité) et colorimétriques (mean R, G, B) montrent des différences nettes entre les classes saines (bleu) et malades (rouge). Les plantes malades présentent souvent une aire et une circularité plus faibles, avec des valeurs de teinte (H) et saturation (S) distinctes. Les mesures de texture (hog, fft, hu) confirment cette séparation.

#### Risques :

Les plantes avec ces caractéristiques tendent vers une altération de structure et de couleur indicative de stress ou infection. Une mauvaise identification pourrait survenir si certaines valeurs se chevauchent, surtout dans les variables de texture.

## Conclusions :

Le profil global du graphique correspond davantage aux distributions observées pour les plantes malades. Les écarts sur la couleur et la morphologie suggèrent un état non sain. Cette plante est donc probablement malade.

## **Objectif3 : Quelle est cette maladie ?**

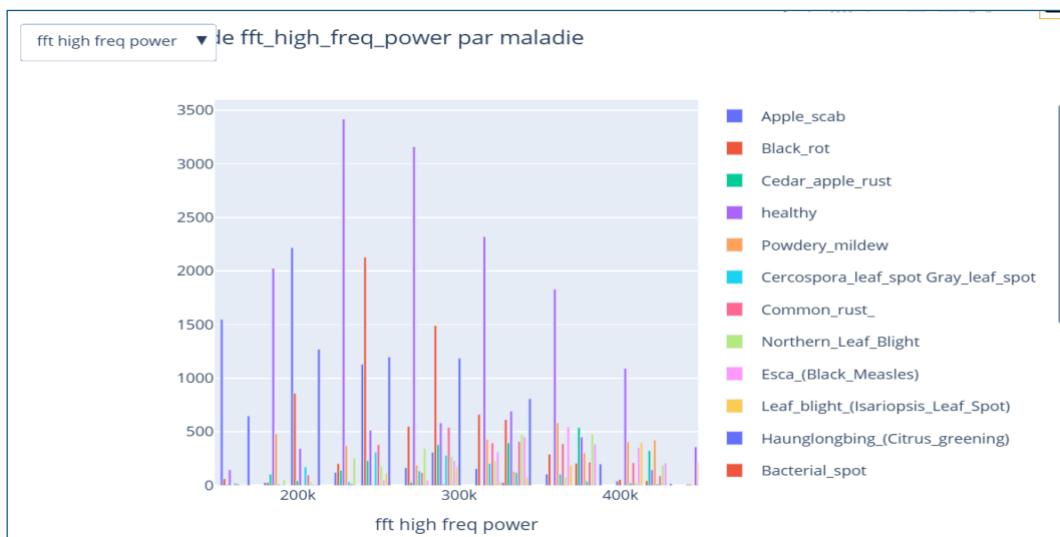


Figure : Distribution des caractéristiques pour objectif 3 "Quelle est la maladie ?"

## Observations :

Les graphiques présentent la répartition des images par maladie, toutes plantes confondues. On constate que certaines maladies (ex : Apple\_scab, Powdery\_mildew, Leaf\_blight\_, Haunglongbing\_) sont beaucoup plus représentées dans le jeu de données, tandis que d'autres ne comptent que quelques exemples. Ce déséquilibre entre maladies est très marqué, certaines pathologies rares n'apparaissant que sur une poignée d'images.

## Risques :

Un fort déséquilibre entre maladies expose le modèle à un risque de biais : il apprendra surtout à reconnaître les maladies majoritaires et aura du mal à prédire correctement les maladies rares, même si elles sont importantes en pratique.

## Conclusion :

Pour répondre correctement à l'objectif 3 (identifier la maladie), il sera essentiel de corriger ou d'atténuer ce déséquilibre (ex : data augmentation, sous-échantillonnage, pondérations lors de l'apprentissage). Cela garantira que le modèle accorde une attention équitable à chaque maladie et que la détection des maladies rares soit suffisamment fiable pour un usage terrain.

## **Représentations visuelles des caractéristiques :**

Ces visualisations permettent d'identifier des variables clés qui séparent bien les plantes saines des malades. Elles permettent également de repérer les comportements atypiques (valeurs extrêmes) qui méritent une investigation (plante en transition de santé, mesure erronée, etc.). Aider au choix des variables pour un futur modèle de classification visant à prédire si une plante est malade. Le nom du dossier contenant les fichiers HTML des Boxplots : Boxplots

- **Boxplot1 et Boxplot2** : ils présentent une grande quantité de données et plusieurs classes, mais la lisibilité est réduite : difficile d'identifier visuellement des différences marquées entre malades et non-malades.
- **Boxplot3** : intéressant car il met en évidence des médianes différentes entre maladies et une variabilité interne bien visible, mais il est légèrement moins clair que Boxplot2 pour distinguer les distributions.
- **Boxplot4** : plus simple et lisible, mais les différences entre classes sont moins marquées, limitant son intérêt pour identifier une plante malade.
- **Boxplot5** : il met en évidence de fortes dispersions avec des valeurs extrêmes (outliers), mais sans séparation claire par maladie.
- **Boxplot6** : comparable au Boxplot3 en termes de clarté, il offre une bonne visualisation de la dispersion mais moins de contraste entre classes.
- **Boxplot7** : très riche en points et en variabilité, mais les différences entre classes sont moins nettes, rendant l'interprétation plus complexe.
- **Boxplot8** : plus condensé visuellement et facile à lire, mais la séparation des distributions entre maladies y est peu marquée.

Explorons un des boxplots : le **Boxplot2** particulièrement intéressant à commenter pour l'objectif 3 : “*Quelle est cette maladie ?*”.

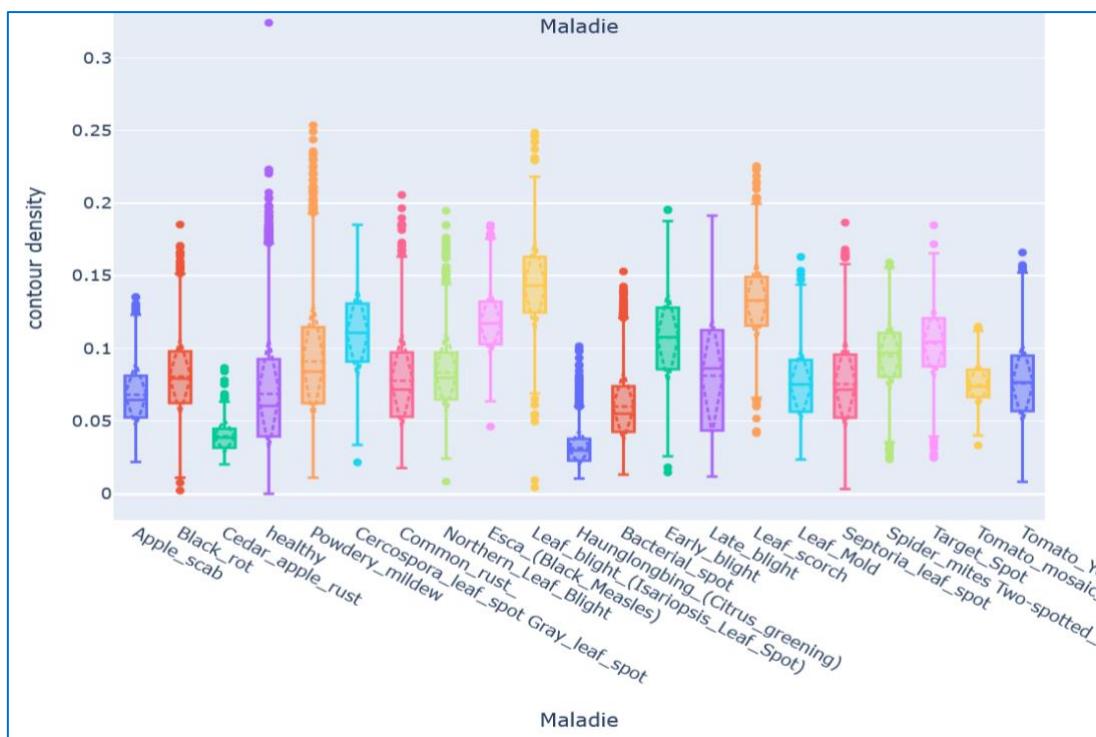


Figure : Boxplots de la densité de contours selon les différentes maladies

Le Boxplot2 met en évidence la variable contour\_density pour distinguer deux maladies du pommier : Apple\_scab et Black\_rot. Ces maladies présentent des symptômes visuellement contrastés sur un même type de feuilles de pommier, limitant les biais liés aux différences d'espèces et facilitant l'interprétation des écarts de médiane et de dispersion. Cette comparaison illustre clairement la capacité des boxplots à révéler des variables discriminantes pour l'objectif 2, qui vise à isoler les caractéristiques clés selon le type de maladie. Un axe d'amélioration consistera à générer des boxplots par plante, afin d'explorer d'autres maladies (ex. Leaf\_blight ou Haunglongbing) tout en conservant la cohérence inter-espèces.

Histogramme présentant la distribution des caractéristiques par type de plantes : Le nom du fichier HTML : distributions\_features\_targets.html

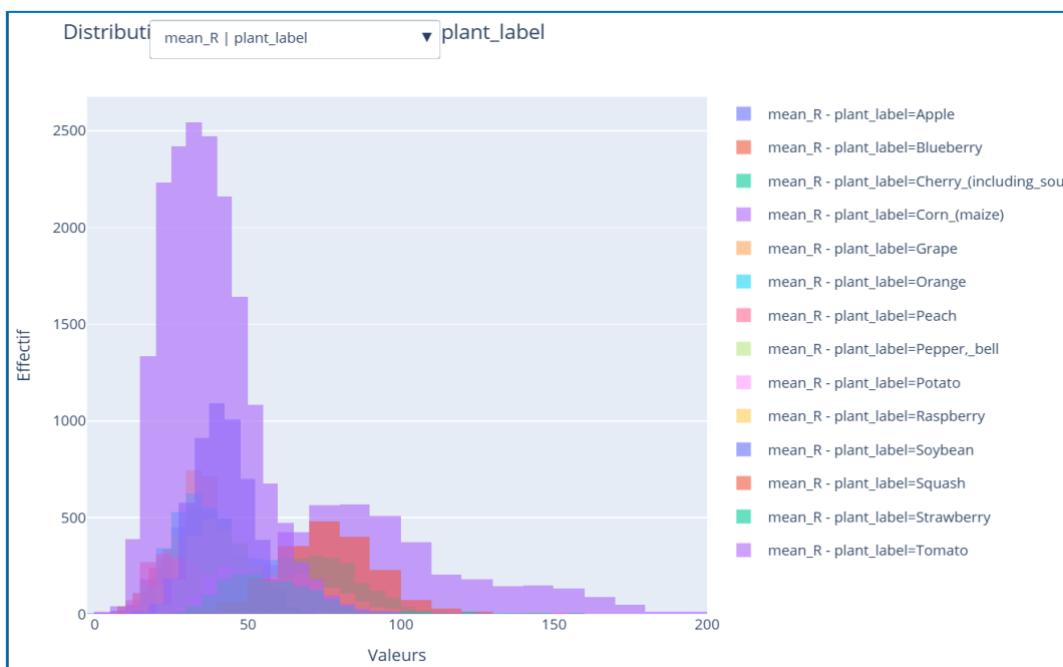


Figure : Histogramme de la distribution des caractéristiques par type de plantes

### Relations entre les caractéristiques

Heatmaps présentant les corrélations entre features: Nom du fichier HTML:

heatmap\_features\_grouped.html

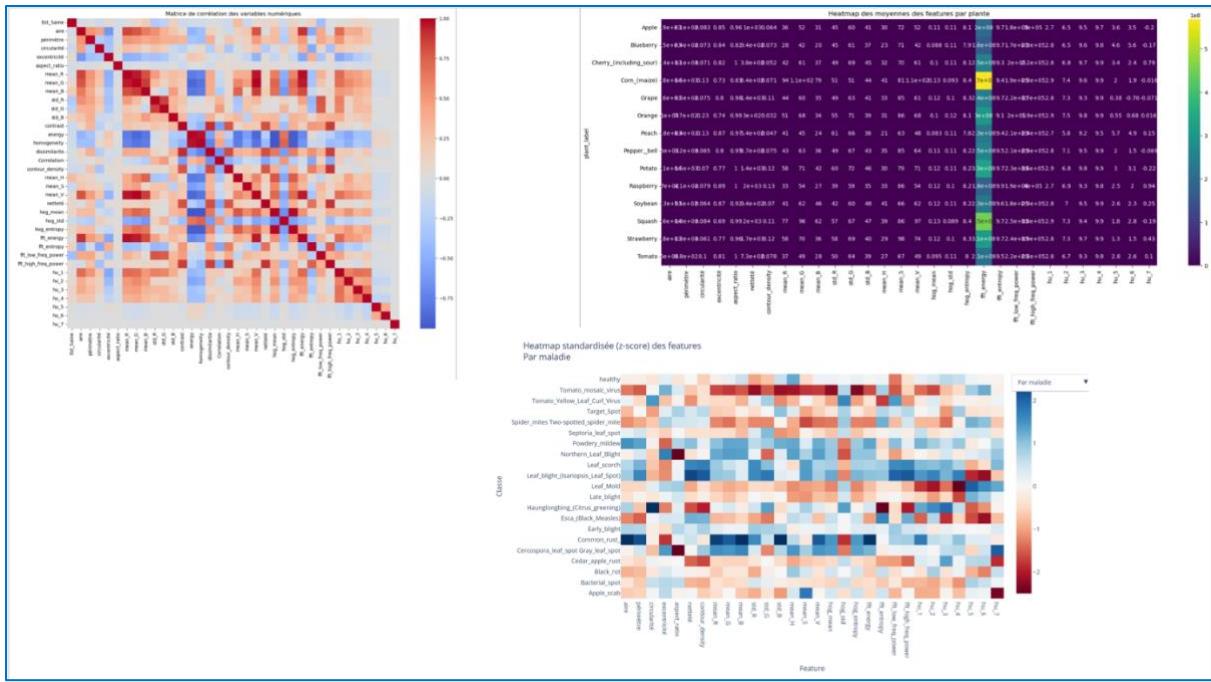


Figure : Heatmaps des corrélations entre features

### Observations :

- On observe une grande variabilité des profils z-score des features entre les différentes espèces de plantes.
- Certaines plantes présentent des profils très contrastés (zones rouges/bleues marquées), ce qui indique des caractéristiques spécifiques fortes.
- Aucune plante n'a exactement la même "empreinte" de feature, c'est intéressant pour la classification d'espèces.
- Le contraste est beaucoup plus net : chaque feature est soit nettement au-dessus, soit nettement au-dessous de la moyenne globale.
- La plupart des features discriminent de façon binaire entre sain et malade : fort potentiel pour des modèles de classification binaire.
- Il existe des patterns différenciés entre maladies, avec des features fortement déviées pour certaines pathologies.
- Les profils "healthy" (sains) se distinguent assez bien, avec des scores souvent proches de la moyenne (blancs).
- Certaines maladies affichent des signatures très marquées sur certaines features, ce qui peut aider à leur identification automatique.

### Risques :

- Certains groupes de features semblent évoluer ensemble. A priori, il y a de la redondance. Risque de surapprentissage
- Variabilité intra-classe : Pour certaines plantes ou maladies, la couleur des cases oscille beaucoup, ce qui peut indiquer une hétérogénéité importante dans la classe. Cela pourrait compliquer la classification fine.
- Effet de standardisation : Comme l'échelle est relative (z-score), une forte déviation sur une feature peut être causée soit par une vraie différence, soit par une variance très faible pour cette feature.

- Binarité excessive (statut sain/malade) : Si les différences sont trop franches et systématiques sur toutes les features, risque d'avoir des modèles "trop parfaits" sur le jeu d'entraînement et moins robustes en situation réelle

### Conclusions :

- Sélection de features pertinentes : Utiliser l'information des heatmaps pour prioriser les features qui montrent des différences nettes entre classes (plante, maladie, statut) tout en surveillant les redondances.
- Analyse approfondie des profils atypiques : Vérifier pourquoi certaines plantes ou maladies ont des patterns très différents ou très fluctuants, et si cela correspond à des outliers, ou une vraie diversité biologique.
- Affinement des modèles :
  - Pour la classification binaire (sain/malade), de nombreuses features sont prometteuses : tester différents algorithmes.
  - Pour la classification multi-classes (espèce, maladie), travailler la réduction de dimensionnalité et l'ingénierie de features pour optimiser la séparation entre groupes.

Nous n'avons pas eu le temps d'utiliser l'Analyse en Composantes Principales (PCA). Pour la prochaine itération, il sera pertinent de réaliser une PCA sur l'ensemble des variables quantitatives extraites, après standardisation, afin de visualiser la structure globale des données. Cette analyse permettra d'explorer la capacité des features à séparer les plantes selon leur maladie ou leur état de santé via des scatterplots des deux premières composantes principales. Enfin, il sera intéressant de reproduire cette approche séparément par espèce pour mieux comprendre la distinction entre maladies au sein d'une même plante.

## 6.2. Les méthodes pour analyser l'importance des features

Voici un tableau de synthèse des différentes approches possible pour analyser l'importance des features.

Méthode	Type	Principe	Classification binaire	Classification multi-classes	Avantages	Inconvénients
Variance Threshold	Filtre	Supprime les features dont la variance est inférieure à un seuil	✓	✓	Très rapide à exécuter	Ne tient pas compte de la cible
Corrélation (Pearson)	Filtre	Retire les features fortement corrélées entre elles	✓	✓	Réduit multicolinéarité	Seulement linéaire
Chi-2	Filtre	Test d'indépendance entre feature catégorielle et cible	✓	Limitée (binaires ou k-classes >2)	Simple et efficace pour variables discrètes	Ne gère pas bien les variables continues
ANOVA F-test	Filtre	Compare moyennes des groupes de la cible	-	✓	Approprié pour variables numériques	Hypothèses normales
Mutual Information	Filtre	Mesure l'information partagée entre feature et cible	✓	✓	Capture relations non linéaires	Plus coûteux qu'un test univarié
Recursive Feature Elimination	Wrapper	Enlève itérativement la moins importante via un estimateur	✓	✓	Prend en compte l'interaction features-cible	Lourd en calcul
Sélection séquentielle (SFS/SBS)	Wrapper	Ajoute ou retire séquentiellement les meilleures features	✓	✓	Flexible (forward/backward)	Risque de surapprentissage, coûteux
L1 Regularization (Lasso)	Intégré	Pénalise les coefficients pour en annuler certains	✓	✓ (via "one-vs-rest")	Rapide, intégré au modèle	Seuil dépendant du paramètre de pénalité
Importance d'arbre (Random Forest)	Intégré	Mesure la réduction d'impureté apportée par chaque feature	✓	✓	Gère variables mixtes, robuste au bruit	Biais vers variables à plus de niveaux
Gradient Boosting Importances	Intégré	Comme pour les forêts, mais via boosting	✓	✓	Meilleure précision souvent	Plus lent que Random Forest

Tableau de synthèse des différentes approches possible pour analyser l'importance des features

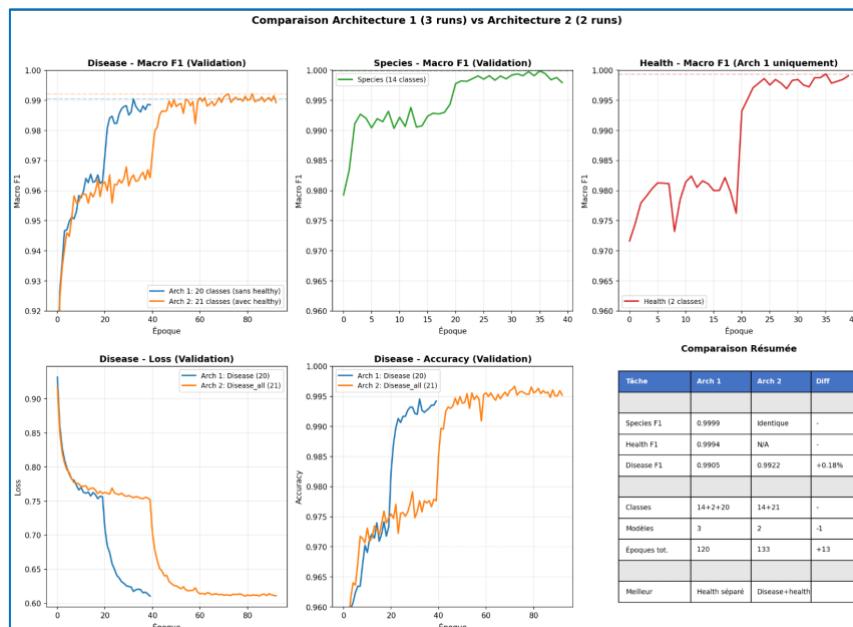
- Filtre = sélection avant apprentissage, indépendante du modèle.
- Wrapper = sélection autour du modèle, itérative et évaluée par performance.
- Intégré = sélection pendant l'apprentissage, via des pénalités ou des métriques internes au modèle

### 6.3. Comparaison des caractéristiques des modèles ML

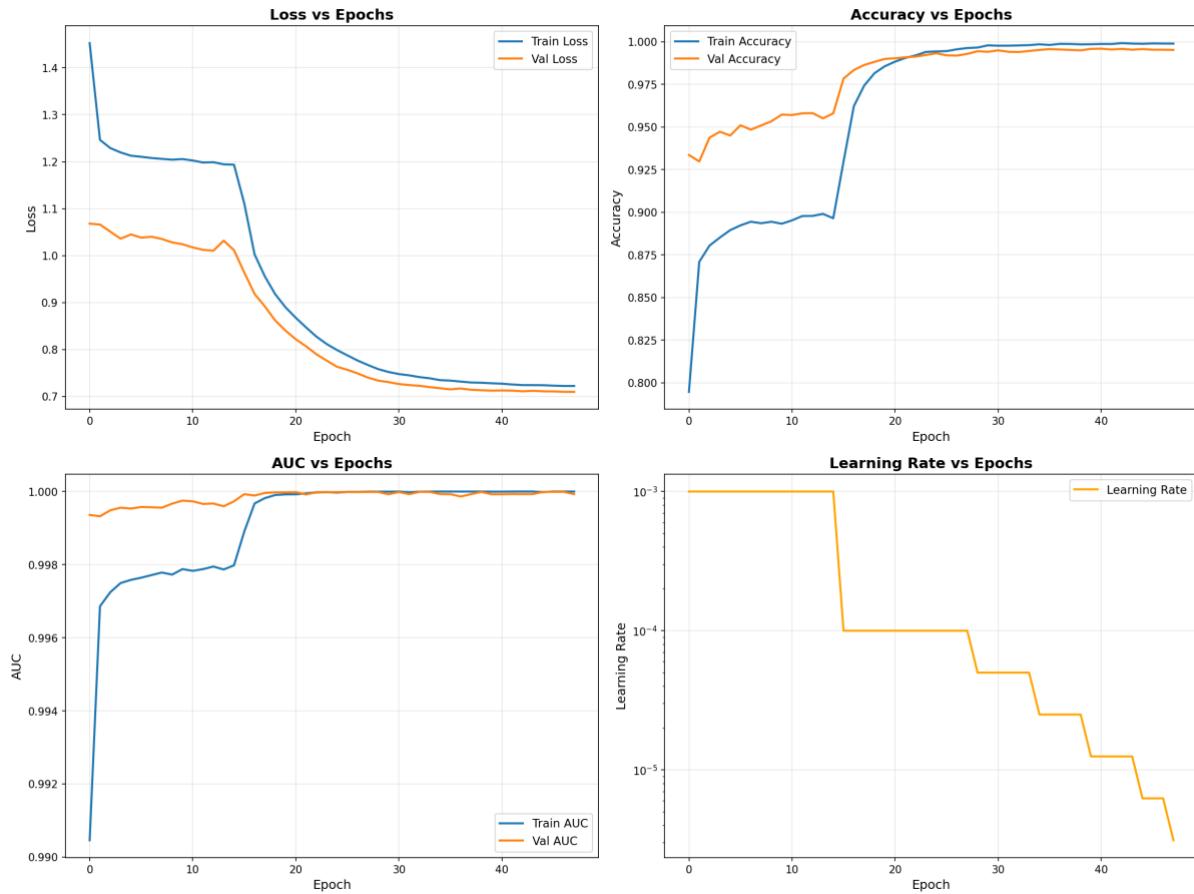
Caractéristique	SVM RBF	XGBoost	Extra Trees	Logistic Regression
Script	models/svm_rbf_plantvillage_features_selected_baseline.py	models/xgboost_baseline.py	models/Xtra_trees.py	models/Log_reg.py
Modèle	SVC(kernel="rbf", decision_function_shape="ovr")	XGBClassifier (GPU si dispo, device="cuda")	ExtraTreesClassifier	LogisticRegression (multinomial, lbfgs)
Pipeline	RobustScaler → SVC	RobustScaler → XGB	RobustScaler → ExtraTrees	RobustScaler → LogisticRegression
CSV (features)	dataset/plantVillage/csv/clean_data_plantvillage_segmented_all_with_features.csv			
Cible	species	species (encodée via LabelEncoder)	species	species
Grille hyperparamètres (résumé)	svm__C=[76.0]; svm__class_weight=[None, "balanced"]	xgb__n_estimators=[300]; xgb__max_depth=[6]; xgb__learning_rate=[0.1,0.05]; xgb__subsample=[0.8]; xgb__colsample_bytree=[0.8]	xtr__n_estimators=[300]; xtr__class_weight=[None, "balanced"]	logreg__C=[0.1, 1.0, 10.0]; logreg__class_weight=[None, "balanced"]
CV (GridSearchCV)	StratifiedKFold(n_splits=5, shuffle=True, random_state=42)	StratifiedKFold(n_splits=5, shuffle=True, random_state=42)	StratifiedKFold(n_splits=5, shuffle=True, random_state=42)	StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
Scoring / refit	{"balanced_accuracy", "f1_macro"}; refit="f1_macro"	{"balanced_accuracy", "f1_macro"}; refit="f1_macro"	{"balanced_accuracy", "f1_macro"}; refit="f1_macro"	{"balanced_accuracy", "f1_macro"}; refit="f1_macro"
n_jobs (CV)	-1	1 si GPU, sinon -1	1	1
Courbes de validation	C, gamma	n_estimators, max_depth	n_estimators, max_features	C
Repeated CV (seeds)	[1,2,3,4,5] (n_jobs=-1)	[1,2,3] (n_jobs=1 si GPU, sinon -1)	[1,2,3] (n_jobs=1)	[1,2,3] (n_jobs=1)
Bootstrap (B)	1000	500	500	500
Mesure(s) de complexité	Ratio vecteurs de support	n_estimators, max_depth, n_classes	n_estimators, profondeur moyenne	n_classes, n_features,   coef   <sub>2</sub>
Artefacts principaux	rappor, cm.csv/html, cm_interactive, galerie erreurs, matrice interactive avec images	rappor, cm.csv/html, cm_interactive, label_mapping.json	rappor, cm.csv/html, cm_interactive	rappor, cm.csv/html, cm_interactive
Dossier résultats	results_modifiés/models/svm_rbf_basel ine_features_selected/	results_modifiés/models/xgb_baseli ne/	results_modifiés/models/extra_tree	results_modifiés/models/logre g_baseline/

### 6.4. Métriques trackées des architectures DL

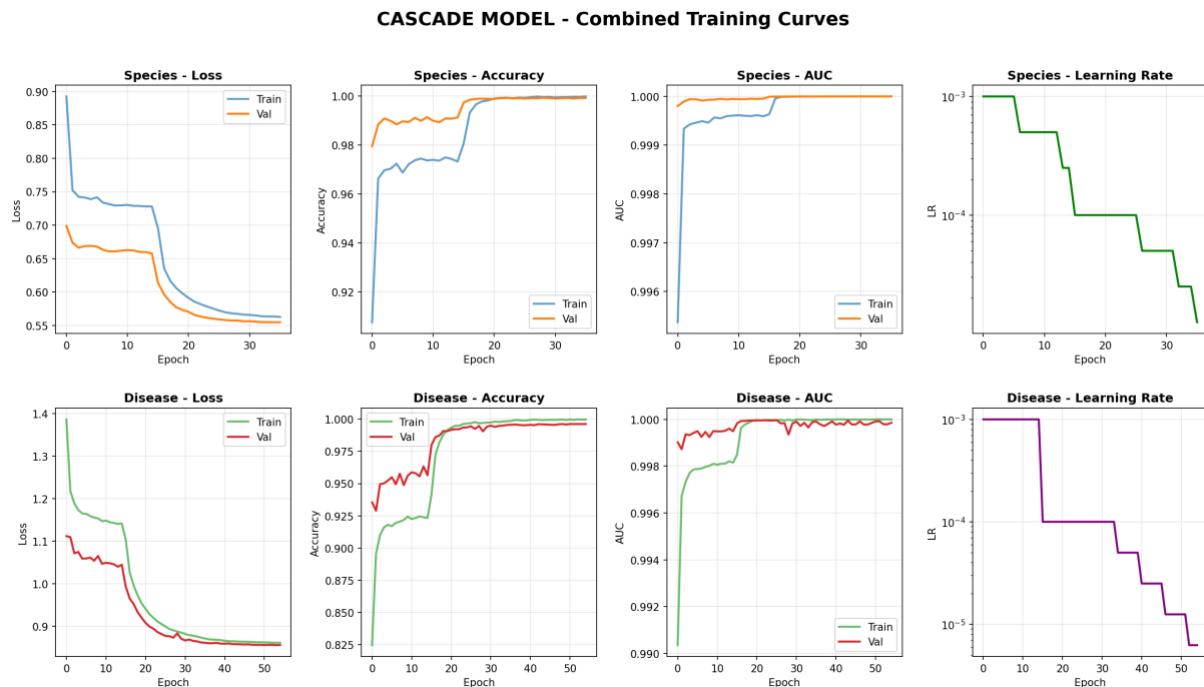
#### Architecture 1 et Architecture 2 :



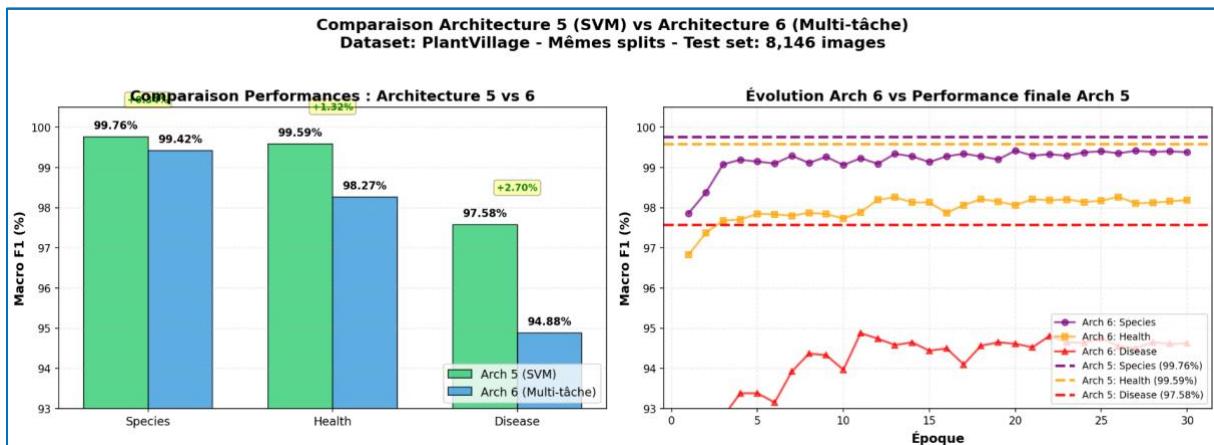
### Architecture 3 :



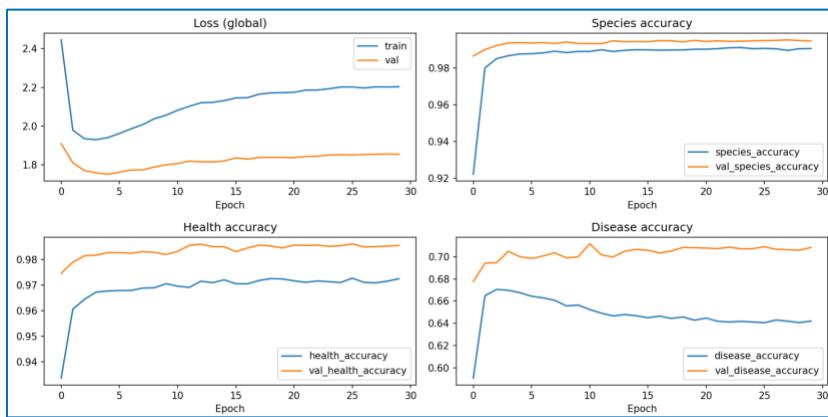
### Architecture 4 :



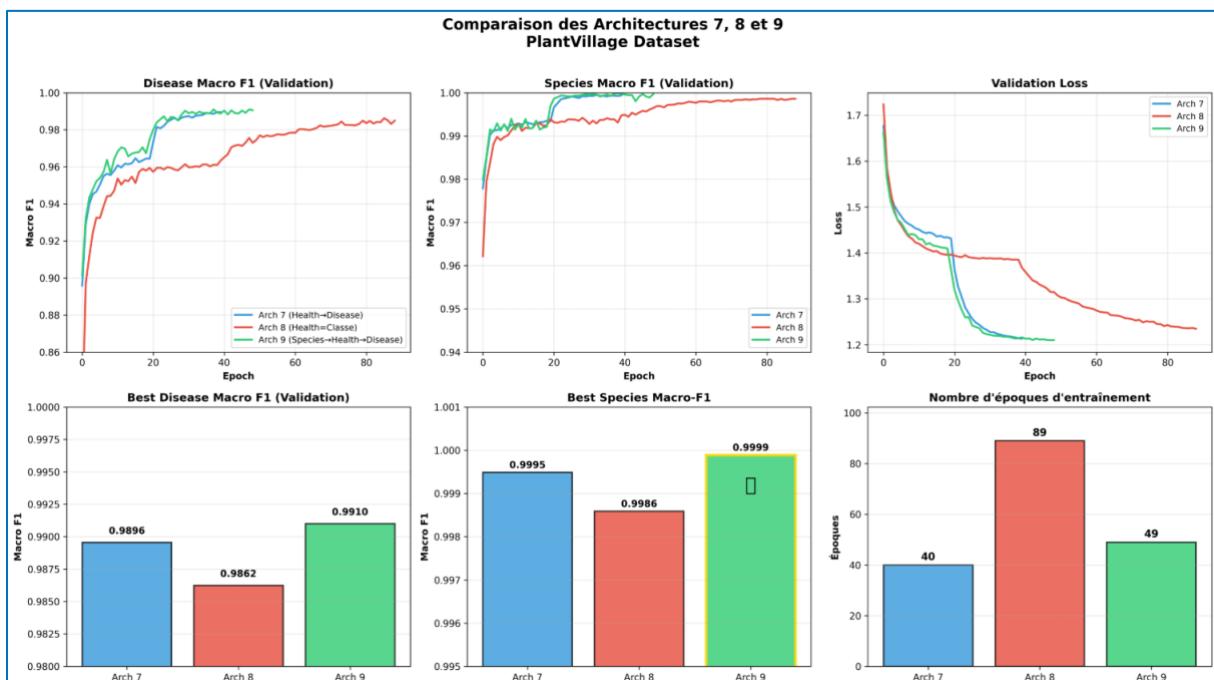
## Architecture 5 et architecture 6 :



## Architecture 6 :



## Architecture 7, Architecture 8 et Architecture 9 :



## 6.5. Caractéristiques des features

N° Col	Nom de la colonne	Description	Distribution des valeurs	Librairie / Source	Fonction
0	is_black	flag indiquant si l'image est totalement noire (1) ou non (0)	Majorité à 0	OpenCV / NumPy	Si l'image est globalement noire ou inutilisable (bruit, absence de données). Seut au filtrage.
7	dimension	Taille de la boîte englobante de la feuille	Continue, valeurs positives	OpenCV	Taille de l'image ou de la région d'intérêt.
8	aire	Surface de la feuille détectée via contour (pixel) <sup>2</sup>	Corrélée à dimension	OpenCV	Surface occupée par la tache ou la plante détectée.
9	périmètre	Longueur du contour de la feuille (pixel)	Corrélée à aire	OpenCV	Longueur du contour de la forme détectée.
10	circularité	$4 \pi \text{aire} / (\text{périmètre}^2)$ : 1 = parfait cercle (Dimensionless)	Entre 0 et 1	OpenCV	Mesure de la ressemblance de la forme à un cercle : 1 pour un cercle parfait.
11	excentricité	Ratio excentricité de la région (0 = cercle, 1 = ligne) (Dimensionless)	Entre 0 et 1	OpenCV	Mesure de l'élongation (distance entre les foyers d'une ellipse).
12	aspect_ratio	Rapport largeur / hauteur de la boîte englobante (Dimensionless)	Valeurs > 0	OpenCV	Rapport largeur / hauteur de la forme ; indique si elle est étirée.
13	mean_R	Moyenne du canal Rouge de la feuille (Intensité normalisée (0–1)	Entre 0 et 1	OpenCV	Moyenne des intensités sur les canaux Rouge, Vert, Bleu. Donne la teinte dominante.
14	mean_G	Moyenne du canal Vert de la feuille (Intensité normalisée (0–1)	Entre 0 et 1	OpenCV	Moyenne des intensités sur les canaux Rouge, Vert, Bleu. Donne la teinte dominante.
15	mean_B	Moyenne du canal Bleu de la feuille (Intensité normalisée (0–1)	Entre 0 et 1	OpenCV	Moyenne des intensités sur les canaux Rouge, Vert, Bleu. Donne la teinte dominante.
16	std_R	Écart-type du canal Rouge (Intensité normalisée (0–1)	Entre 0 et 1	OpenCV / NumPy	Variation (écart-type) des couleurs — mesure la diversité colorimétrique.
17	std_G	Écart-type du canal Vert (Intensité normalisée (0–1)	Entre 0 et 1	OpenCV / NumPy	Variation (écart-type) des couleurs — mesure la diversité colorimétrique.
18	std_B	Écart-type du canal Bleu (Intensité normalisée (0–1)	Entre 0 et 1	OpenCV / NumPy	Variation (écart-type) des couleurs — mesure la diversité colorimétrique.
19	mean_H	Moyenne du canal Teinte (Hue) Degrés (0–179)	0–179 degrés	OpenCV	Moyennes dans l'espace HSV (Teinte, Saturation, Valeur) : plus robustes aux variations de lumière.
20	mean_S	Moyenne du canal Saturation Intensité (0–255)	0–255	OpenCV	Moyennes dans l'espace HSV (Teinte, Saturation, Valeur) : plus robustes aux variations de lumière.
21	mean_V	Moyenne du canal Valeur (brightness) Intensité (0–255)	0–255	OpenCV	Moyennes dans l'espace HSV (Teinte, Saturation, Valeur) : plus robustes aux variations de lumière.
22	netteté	Variance du Laplacien (sharpness measure) Intensité <sup>2</sup> (dimensionless)	Valeurs positives	OpenCV (Laplacian)	Mesure de la clarté des détails dans l'image (souvent calculée par des gradients ou Laplacien).
23	contrast	GLCM contrast : mesure de la variation locale de gris (Dimensionless)	Valeurs positives	skimage.feature.greycomatrix	Différence locale d'intensité : élevé si beaucoup de variations entre pixels voisins.
24	energy	GLCM energy : somme des carrés de la matrice de cooccurrence (Dimensionless)	0–1	skimage.feature.greycomatrix	Homogénéité globale (somme des carrés des probabilités dans la GLCM).
25	homogéneity	GLCM homogeneity : voisinage uniforme vs varié (Dimensionless)	0–1	skimage.feature.greycomatrix	Inverse de la variance locale : élevé si les pixels voisins ont des valeurs proches.
26	dissimilarité	GLCM dissimilarity : somme des écarts pondérés (Dimensionless)	Valeurs positives	skimage.feature.greycomatrix	Mesure brute des écarts entre pixels voisins.
27	Correlation	GLCM correlation : dépendance linéaire des intensités voisines (Dimensionless)	-1 à 1	skimage.feature.greycomatrix	Corrélation entre intensités de pixels : détecte les motifs réguliers ou structurés.
28	contour_density	Densité du contour = périmètre / aire (Pixels <sup>-1</sup> ) (dimensionless)	périmètre / aire	OpenCV	Densité des bords détectés ; élevé si beaucoup de transitions nettes (formes, nervures).
29	hu_1	7 Moments de Hu invariants (shape descriptors invariants) (Dimensionless)	Valeurs continues	OpenCV (Hu Moments)	Représente la variance (dispersion) globale. Sensible à la structure générale.
30	hu_2	7 Moments de Hu invariants (shape descriptors invariants) (Dimensionless)	Valeurs continues	OpenCV (Hu Moments)	Différencie les formes symétriques des asymétriques.
31	hu_3	7 Moments de Hu invariants (shape descriptors invariants) (Dimensionless)	Valeurs continues	OpenCV (Hu Moments)	Capte la régularité des contours (peut indiquer des anomalies locales).
32	hu_4	7 Moments de Hu invariants (shape descriptors invariants) (Dimensionless)	Valeurs continues	OpenCV (Hu Moments)	Sensible à l'asymétrie diagonale.
33	hu_5	7 Moments de Hu invariants (shape descriptors invariants) (Dimensionless)	Valeurs continues	OpenCV (Hu Moments)	Capte la torsion ou la déformation de la forme.
34	hu_6	7 Moments de Hu invariants (shape descriptors invariants) (Dimensionless)	Valeurs continues	OpenCV (Hu Moments)	Très sensible aux petites irrégularités — utile pour distinguer des textures fines.
35	hu_7	7 Moments de Hu invariants (shape descriptors invariants) (Dimensionless)	Valeurs continues	OpenCV (Hu Moments)	Compte plusieurs effets de symétrie ; utilisé pour distinguer des formes très similaires.
36	hog_mean	Moyenne des valeurs HOG (Histogramme des gradients orientés)	Valeurs positives	scikit-image (HOG)	Capture la variation locale des orientations de bords — reflète les textures et contours répétitifs.
37	hog_std	Écart-type des valeurs HOG	Valeurs positives	scikit-image (HOG)	Moyenne ou histogramme global des orientations — donne un résumé des formes présentes.
38	hog_entropy	Entropie des histogrammes HOG	Valeurs positives	scikit-image (HOG)	L'entropie des gradients (HOG) indique la diversité directionnelle — utile pour détecter des irrégularités.
39	fft_energy	Énergie spectrale de la transformée de Fourier	Valeurs positives	numpy.fft	Indique l'intensité globale des motifs périodiques — utile pour les textures régulières.
40	fft_entropy	Entropie spectrale (Transformée de Fourier)	Valeurs positives	numpy.fft	Mesurée par l'entropie du spectre — reflète la diversité des composantes fréquentielles.
41	fft_low_freq_power	Puissance dans la bande de basse fréquence (Transformée de Fourier)	Valeurs positives	numpy.fft (basses fréquences)	Concentration des basses fréquences — traduit les grandes structures de l'image.
42	fft_high_freq_power	Puissance dans la bande de haute fréquence (Transformée de Fourier)	Valeurs positives	numpy.fft (hautes fréquences)	Concentration des hautes fréquences — capte les contours nets et les détails fins.