République Algérienne Démocratique et Populaire
الجمهورية الجزائرية الديمقراطية الشعبية
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique
وزارة التعليم العالي و البحث العلمي
École nationale Polytechnique

Département électronique

Mémoire de projet de fin d'études

Pour l'obtention du diplôme d'Ingénieur d'État en Électronique

Conception d'une interface Web pour le diagnostic des maladies des plantes par apprentissage profond

Salah Eddine DAHMANI

Sous la direction de Mme. Nesrine Bouadjenek Dr. ENP, Alger

Présenté et soutenu publiquement le 04/07/2022 auprès des membres du jury :

Président	M. Mourad	Adnane	Prof.	ENP, Alger
Promotrice	Mme. Nesrine	Bouadjenek	Dr.	ENP, Alger
Examinateur	M. Hicham	Bousbia-Salah	Prof.	ENP, Alger

ENP 2022

10, Avenue des Frères Oudek, Hassen Badi, BP. 182, 16200 El Harrach, Alger, Algérie.
www.enp.edu.dz

République Algérienne Démocratique et Populaire
الجمهورية الجزائرية الديمقراطية الشعبية
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique
وزارة التعليم العالي و البحث العلمي
École nationale Polytechnique

Département électronique

Mémoire de projet de fin d'études

Pour l'obtention du diplôme d'Ingénieur d'État en Électronique

Conception d'une interface Web pour le diagnostic des maladies des plantes par apprentissage profond

Salah Eddine DAHMANI

Sous la direction de Mme. Nesrine Bouadjenek Dr. ENP, Alger

Présenté et soutenu publiquement le 04/07/2022 auprès des membres du jury :

Président	M. Mourad	Adnane	Prof.	ENP, Alger
Promotrice	Mme. Nesrine	Bouadjenek	Dr.	ENP, Alger
Examinateur	M. Hicham	Bousbia-Salah	Prof.	ENP, Alger

ENP 2022

10, Avenue des Frères Oudek, Hassen Badi, BP. 182, 16200 El Harrach, Alger, Algérie.
www.enp.edu.dz

Dédicace

“

À ma chère Maman, à ma famille et à mes amis.

À tous ceux qui me sont chers, à vous tous

Merci.

”

- Salah Eddine

Remerciements

Je tiens à exprimer ma sincère gratitude à ma promotrice, Dr. Nesrine BOUADJENEK, qui m'a soutenu tout au long de ce travail. Je la remercie pour sa patience, sa motivation, sa disponibilité et son dévouement sans pareil. je n'aurais pas pu espérer un meilleur superviseur et mentor.

Je remercie tout particulièrement les membres du jury qui ont accepté de consacrer leur temps à examiner ce travail : Monsieur Mourad ADNANE, Professeur à l'Ecole Nationale Polytechnique et Monsieur Hicham BOUSBIA-SALAH, Professeur à l'Ecole Nationale Polytechnique.

Je tiens à adresser mes remerciements et ma gratitude à tous mes professeurs de l'Ecole Nationale Polytechnique, qui m'ont guidé tout au long de mes cinq années d'études.

Je remercie également tous mes collègues de l'Ecole Nationale Polytechnique, avec qui j'ai partagé de précieux moments.

Enfin, je remercie tous ceux qui m'ont aidé de près ou de loin tout au long de mon cursus académique.

ملخص

تعتبر الأمراض التي تصيب النباتات من الأسباب الخطيرة لتدهور كمية ونوعية الإنتاج مما يؤدي إلى خسائر اقتصادية. وبالتالي، فإن التعرف على الأمراض في النباتات مهم للغاية. تظهر أعراض الأمراض في أجزاء مختلفة من النباتات. ومع ذلك، فإن الأوراق هي الأكثر شيوعًا للكشف عن العدوى. يستخدم العديد من الباحثين تقنيات الرؤية الحاسوبية لاكتشاف الأمراض باستخدام صور الأوراق، تقوم دراستنا بتشخيص أمراض النبات باستخدام طريقة الشبكة العصبية العميقة (DNN) بناءً على أعراض المرحلة المبكرة هذه. تم استخدام العديد من نماذج الشبكات العصبية التلافيفية (CNN) مثل AlexNet و VGG16 و ResNet بالإضافة إلى نموذج سنقترحه لاحقًا لتحديد 17 فئة بها 14 مرضًا. بعد ذلك، قمنا ببناء واجهة على شبكة الإنترنت لتشخيص هذه الأمراض باستخدام أحد هذه النماذج.

كلمات مفتاحية : الرؤية الحاسوبية، تشخيص أمراض النبات، .CNN

Abstract

Degradation in the quantity and quality of production leads to economic losses. Thus, recognition of plant diseases is very important. Disease symptoms appear in different parts of the plants. However, it is the leaves that are most commonly used to detect infection. Many researchers use computer vision techniques to detect diseases using leaf images. Our study diagnoses plant diseases using the deep neural network (DNN) method based on these early symptoms. Several convolutional neural network (CNN) models such as AlexNet, VGG16 and ResNet were used in addition to a model we will propose later to identify 17 classes with 14 diseases. Then we built a web interface for the diagnosis of these diseases using one of these models.

Keywords : Computer vision, Plant diseases diagnosis ,CNN.

Résumé

La dégradation de la quantité et de la qualité de la production entraîne des pertes économiques. Ainsi, reconnaître les maladies des plantes est très important. Les symptômes de maladies apparaissent dans différentes parties des plantes. Cependant, ce sont les feuilles qui sont le plus couramment utilisées pour détecter l'infection. De nombreux chercheurs utilisent des techniques de vision par ordinateur pour détecter les maladies à l'aide d'images de feuilles. Notre étude diagnostique les maladies des plantes à l'aide de la méthode des réseaux de neurones profonds (DNN) basée sur ces symptômes précoces. Plusieurs modèles de réseaux de neurones convolutifs (CNN) tels que AlexNet, VGG16 et ResNet ont été utilisés en plus d'un modèle que nous proposerons plus tard pour identifier 17 classes avec 14 maladies. Ensuite, nous avons construit une interface Web pour le diagnostic de ces maladies en utilisant l'un de ces modèles.

Mots clés : Vision par ordinateur, Diagnostic des maladies des plantes, CNN.

Table des matières

Liste des tableaux

Table des figures

Liste des abréviations

IA *Intelligence artificielle.*

DL *Deep learning.*

CNN *Convolutional neural network.*

DNN *Deep neural network.*

RVB *Rouge Vert Bleu.*

FCN *Fully connected network.*

ACP *Analyse en composantes principales.*

MLP *Multi Layer Perceptron.*

SVM *Support Vector Machine.*

ReLu *Fonction Unité Linéaire Rectifiée.*

SGD *Stochastic gradient descent.*

Adam *Adaptive Moment Estimation.*

GPU *Graphics Processing Unit.*

TPR *True positive rate.*

FPR *False positive rate.*

TNR *True negative rate.*

FNR *False negative rate.*

API *Application Programming Interface.*

VM *Virtual Machine.*

Introduction générale

Les maladies des plantes constituent une menace courante pour la qualité et la quantité de la production agricole mondiale. Les maladies des plantes désastreuses augmentent la pénurie actuelle de produits alimentaires dans laquelle au moins 0,8 milliard de personnes sont insuffisamment nourries [1]. En outre, elles constituent une menace importante pour la sécurité alimentaire, où le nombre de consommateurs augmente chaque jour. Pour réduire les dégâts, il faut identifier le problème immédiatement. En particulier, les maladies des plantes virales n'ont pas de solutions et se propagent rapidement. Ainsi, les plantes infectées doivent être éliminées instantanément pour éviter les infections secondaires. Pour éliminer immédiatement les plantes infectées, le diagnostic de la plante malade est la tâche la plus importante.

Le diagnostic et la détection des maladies des plantes jouent un rôle essentiel pour garantir la qualité et la quantité de la production alimentaire. Le diagnostic automatisé des maladies des plantes est un sujet de recherche inévitable, et il est étudié par différentes approches de chercheurs. Les feuilles des plantes sont l'élément commun de la détection des maladies des plantes, et les symptômes de la plupart des maladies commencent à apparaître sur les feuilles [2]. Par conséquent, l'identification des maladies à l'aide d'images de feuilles est la méthode la plus générale pour les chercheurs. Avec les développements de l'apprentissage automatique, les applications de vision par ordinateur ont connu un énorme succès. Ce succès conduit à la mise en œuvre de nouvelles approches et de nouveaux modèles, qui forment aujourd'hui une nouvelle classe, connue sous le nom d'apprentissage profond. Les techniques d'apprentissage profond ont été introduites dans le domaine agricole et ont gagné en popularité grâce à leur robustesse. Les chercheurs ont créé des réseaux de neurones convolutifs (CNN) qui résolvent le problème de reconnaissance des formes associé aux images.

Les feuilles contiennent des textures et des caractéristiques visuelles qui permettent d'identifier le type de maladie. Par conséquent, l'apprentissage profond avec la vision par ordinateur permet de résoudre ce problème. Ce travail porte en grande partie sur la détermination des maladies des plantes et donc la réduction des pertes de récolte, par conséquent, l'amélioration de l'efficacité de la production. Notre étude diagnostique les maladies des plantes en utilisantdes architectures d'apprentissage profond. Plusieurs modèles CNN ont été utilisés tels que AlexNet, VGG16 et ResNet ainsi qu'un modèle que nous proposerons par la suite, pour identifier 17 classes avec 14 maladies. En dernier, nous avons construit une interface basée sur le web pour faire le diagnostic des maladies des plantes en utilisant l'un de ces modèles.

Ce mémoire est divisé en quatre chapitres qui sont structurés comme suit :

Le premier chapitre : nous introduisons les étapes de traitement qui constituent un système de détection et de classification des plantes malades en allant au-delà des méthodes proposées dans l'état de l'art.

Le deuxième chapitre : on présente un aperçu théorique de l'évolution des réseaux de neurones artificiels vers les réseaux de neurones convolutifs utilisé dans ce travail pour le diagnostic des maladies des plantes.

Le troisième chapitre : on va expliquer la méthodologie de travail suivi, ainsi que l'analyse des performances de différents systèmes mis en œuvre pour l'identification et le diagnostic des maladies des plantes.

Le quatrième chapitre : on va aborder les détails de l'implémentation du modèle d'apprentissage profond dans une interface web, afin de faire le diagnostic des maladies des plantes.

Enfin, nous terminons ce mémoire par une conclusion et quelques perspectives pour un travail futur.

Chapitre 1

État de l'art sur les systèmes de détection et classification des maladies des plantes

1.1 Introduction

L'apparition de maladies végétales a un impact négatif sur la production agricole. Si les maladies des plantes ne sont pas découvertes à temps, l'insécurité alimentaire augmentera [3]. La détection précoce est la base d'une prévention et d'un contrôle efficaces des maladies des plantes, et joue un rôle essentiel dans la gestion et la prise de décision de la production agricole. Ces dernières années, l'identification des maladies des plantes a été une question cruciale.

Les plantes infectées par une maladie présentent généralement des marques ou des lésions évidentes sur les feuilles, les tiges, les fleurs ou les fruits. En général, chaque maladie ou ravageur présente un modèle visible unique qui peut être utilisé pour diagnostiquer les anomalies de manière unique. Habituellement, les feuilles des plantes sont la principale source d'identification des maladies des plantes, et la plupart des symptômes des maladies peuvent commencer à apparaître sur les feuilles [4]. En effet, l'état de santé d'une plante apparaıt dans la couleur, la texture, les bordures, et la taille de ses feuilles. Comme le montre la figure 1.1, tout changement dans ces paramètres, exprime une détérioration de la santé de la plante. Dans la plupart des cas, les experts agricoles et forestiers sont utilisés pour identifier les maladies et les parasites des arbres fruitiers en se basant sur leur expérience. Cette méthode est non seulement subjective, mais aussi longue, laborieuse et inefficace.

Les agriculteurs moins expérimentés risquent de mal juger et d'utiliser des médicaments à l'aveuglette pendant le processus d'identification. La qualité et le rendement entraîneront également une pollution de l'environnement, ce qui provoquera des pertes économiques inutiles. Pour relever ces défis, la recherche sur l'utilisation des techniques de traitement d'images et d'intelligence artificielle pour la détection précoce des maladies des plantes est devenue un sujet de recherche qui connait un grand essor grâce au développement technologiques des outils d'acquisition et de traitement [5].

Fig. 1.1 : Exemples de feuilles représentant des plantes malades - PlantVillage ensemble de données [6]

Dans ce chapitre, nous allons explorer cette méthode de détection des plantes malades. Ainsi, dans ce chapitre, nous introduisons les étapes de traitement qui constituent un système de détection et de classification des plantes malades en allant au-delà des méthodes proposées dans l'état de l'art.

1.2 Maladies des plantes

Les maladies des plantes sont soit causées par un organisme vivant (biotique), soit produites par des impacts environnementaux (abiotiques) tels que la grêle, les gelées de printemps, les conditions météorologiques, les brûlures dues aux produits chimiques, etc. Ces dernières sont moins dangereuses et peuvent être évitées car elles sont non infectieuses et non transmissibles [7]. D'autre part, les maladies biotiques sont les plus dangereuses et causent les plus grands dommages aux cultures. Elles sont classées en trois catégories principales, à savoir :

- **Maladies fongiques :**
  Elles sont causées par des champignons ou des organismes similaires, et sont responsables d'environ 85% des maladies des plantes. Les spores fongiques sont très petites et légères, ce qui signifie qu'elles peuvent se déplacer dans l'air pour infecter d'autres plantes ou arbres.
- **Maladies bactériennes :**
  Elle sont causées par environ 200 types de bactéries, et peuvent se propager par les insectes, les éclaboussures d'eau, d'autres plantes ou outils malades.
- **Maladies virales :**
  Elles sont causées par des virus et sont considérées comme le type le plus rare de maladies des plantes. Cependant, une fois infectées, il n'existe aucun traitement chimique pour éliminer un virus et toutes les plantes suspectes doivent être retirées pour arrêter l'infection. Ils doivent pénétrer physiquement dans la plante et les porteurs les plus courants sont les insectes.

La figure 1.2 indique les 3 types de maladies introduites et les symptômes liés a chaqu'une d'entre elles :

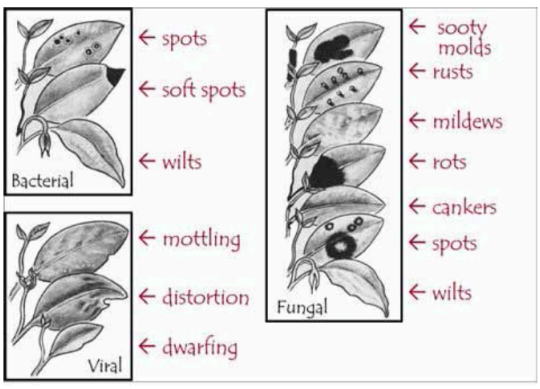


Fig. 1.2 : Différents types de maladies des plantes[8]

1.3 Systèmes de diagnostic des maladies agricoles

Parmi les définitions que nous avons trouvées dans la littérature, celle donnée par [9] décrit succinctement ce qu'est un système de prédiction des maladies agricoles : " C'est un outil de gestion utilisé pour prédire l'apparition ou la détérioration des maladies des plantes cultivées ". Les producteurs utilisent ces systèmes pour prendre des décisions économiques sur les traitements de contrôle des maladies. Les systèmes posent généralement aux producteurs une série de questions sur la sensibilité de la culture hôte et font des recommandations en conjonction avec les conditions météorologiques actuelles et prévisibles. Généralement, les recommandations visent à déterminer le besoin pour le traitement de la maladie associée.

Selon cette définition, la tâche principale d'un système de diagnostic des maladies des plantes est de détecter de manière adéquate l'apparition de maladies à l'avance en passant par plusieurs étapes comme l'indique la figure 1.3, afin que les producteurs puissent prendre des décisions correctes sur l'application de produits phytosanitaires. Dans ce qui suit, nous verrons les bases, les conditions préalables et les processus des systèmes de diagnostic des maladies des plantes.

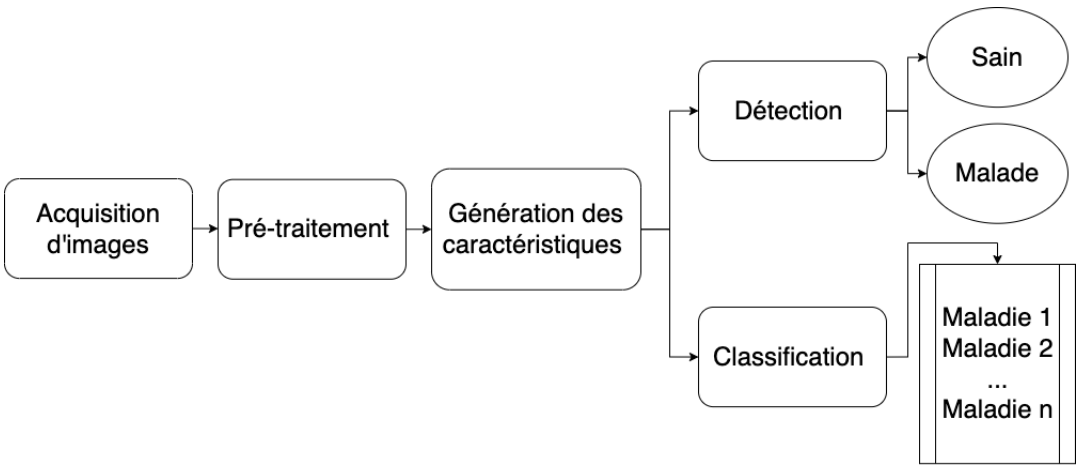


Fig. 1.3 : Schéma synoptique d'un système de diagnostic des maladies des plantes

Caractéristiques du système de diagnostic des maladies agricoles

Selon Lucas et al.,[10] un système de prédiction est dit efficace s'il présente les caractéristiques essentielles suivantes :

- **Fiabilité** : Utiliser des données environnementales fiables, c'est à dire collecter des facteurs climatiques tels que la température ou l'humidité de façon précise.
- **Simplicité** : Le système doit fournir une interface conviviale pour une utilisation facile exploité à grande échelle par les agriculteurs.

- **Importance** : La maladie traitée doit avoir une importance économique sur la culture et être assez sporadique c.-à-d. la possibilité d'un traitement momentané n'est pas applicable.

- **Utilité** : Le système de diagnostic doit être utile c'est à dire, il permet de décharger les producteurs de plusieurs activités de surveillance de la culture en offrant les recommandations nécessaires à l'application des produits chimiques et au contrôle des maladies au moment convenable.

- **Disponibilité** : Les données de l'élément d'interaction doivent être disponibles en temps réel (types de plantes, variétés de plantes, données climatiques, etc.)

- **Rentable** : Le système de diagnostic doit être abordable en termes de coût techniques et de gestion des maladies disponibles.

1.4 Étapes de détection des maladies agricoles

Tout système de vision par ordinateur passe par plusieurs étapes avant d'arriver à la solution souhaitée, comme l'indique le diagramme de la figure 1.4 :

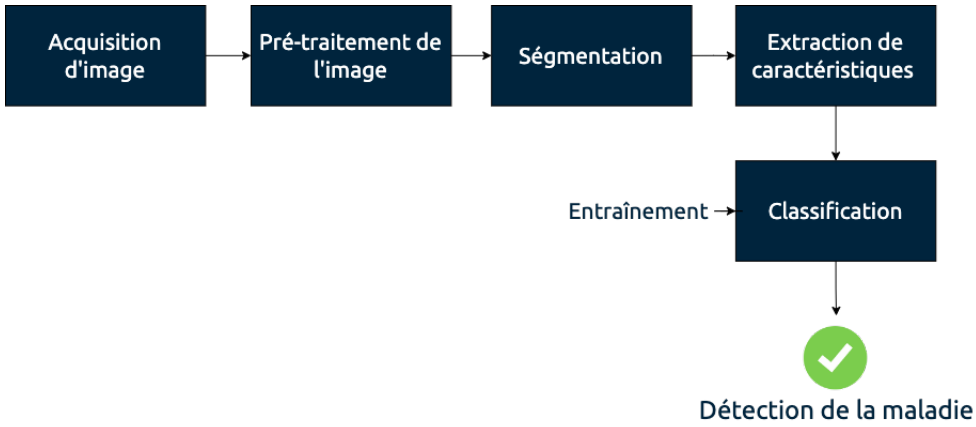


FIG. 1.4 : Schéma synoptique montrant les étapes de détection des maladies agricoles

1.4.1 Collecte de données : Acquisition des images

Comme pour tout système de vision par ordinateur, il s'agit de la première étape. L'acquisition des images est réalisée par différents dispositifs tels que des drones et des caméras placés sur des robots. Plus récemment, les smartphones ont commencé à être utilisés pour développer des applications mobiles de détection des maladies des plantes. Le choix de l'outil d'acquisition dépend de la nature de la zone de contrôle et des objectifs de notre traitement. Pour les zones aux surfaces limitées, comme les serres, les smartphones et les robots suffisent. Cependant, pour de larges surfaces on l'utilisation des drones ou des images satellitaires est recommandée.

1.4.2 Pré-traitement des images

Le prétraitement implique les opérations au plus bas niveau d'abstraction, pour lesquelles l'entrée et la sortie sont uniquement des images [11]. La phase de pré-traitement des images est une étape fondamentale de la vision par ordinateur et du traitement des images. En raison de la dégradation de la qualité de l'image due à la présence d'ombres, de distorsions non spécifiées, de bruit et d'arrière-plans complexes, le prétraitement est l'étape initiale pour améliorer l'image et la rendre appropriée pour un traitement ultérieur [12]. Le plus souvent, dans les principaux ensembles de données, les images sont collectées dans des conditions en temps réel, contenant des informations inappropriées. Ainsi, avant d'extraire les caractéristiques, les images sont prétraitées pour améliorer la précision de calcul du système de détection des maladies des plantes. Le temps de traitement est également réduit en appliquant des opérations de prétraitement comme le redimensionnement et le recadrage.
Les opérations de prétraitement qui sont souvent utilisées, peuvent être résumées comme suit :

(a) **Normalisation des images**
Lorsque la méthode de caractérisation utilisée génère des descripteurs qui dépendent de la taille de l'image, la taille unique doit être normalisée.
Étant donné que les méthodes de classification nécessitent des données de même taille, la normalisation est essentielle. De plus, cette étape peut être utilisée pour réduire la taille des données afin de réduire la complexité du traitement, comme l'indique la figure 1.5 :

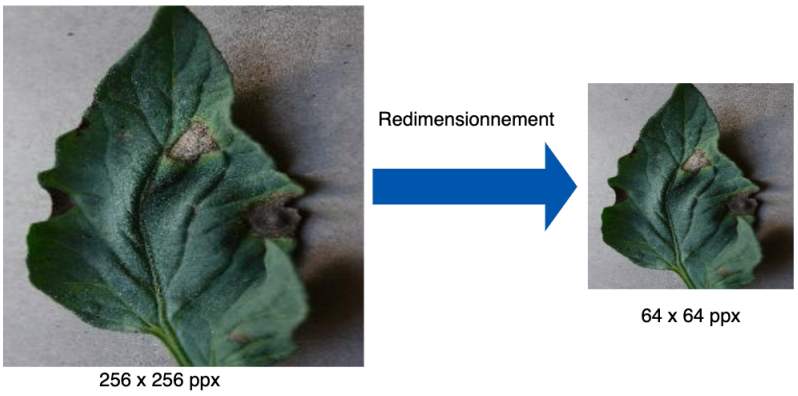


Fig. 1.5 : Redimensionnement d'une image

(b) **Suppression du bruit** Il s'agit de supprimer le bruit liée à la scène, aux conditions de prise de vue ou au capteur d'acquisition par l'application de filtres passe-bas. La figure 1.6 illustre un exemple de filtre gaussien qui permet de supprimer le bruit additif au détriment des détails de contours.

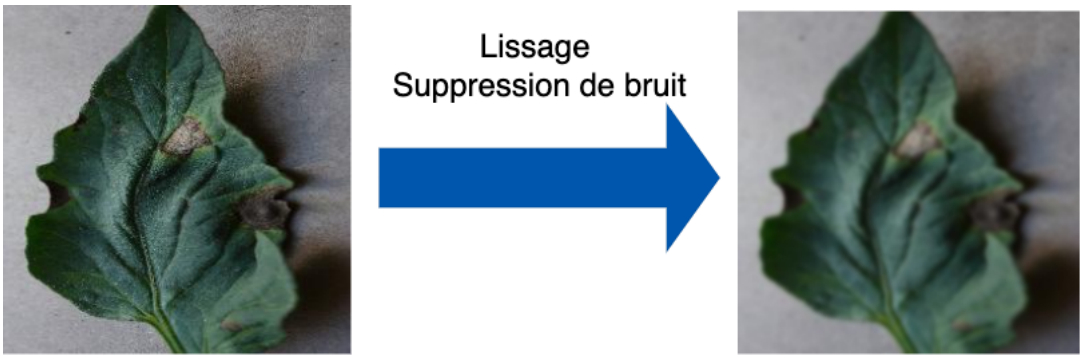


FIG. 1.6 : Lissage d'une image

(c) **Détection de contours**

La détection des contours permet de rehausser la forme d'une feuille qui varie selon l'état de santé des images. Les détecteurs de contours utilisés en vision par ordinateur sont généralement basés sur un filtrage convolutif qui illustre les zones ayant une forte variation des intensités. Dans ce contexte, on distingue les filtres de Sobel, Prewitt, Laplacien, Kirsch, et Canny [13].

A titre illustratif, la figure 1.7 montre la détection des contours par les filtres de Sobel et Laplacien 4 connexes.

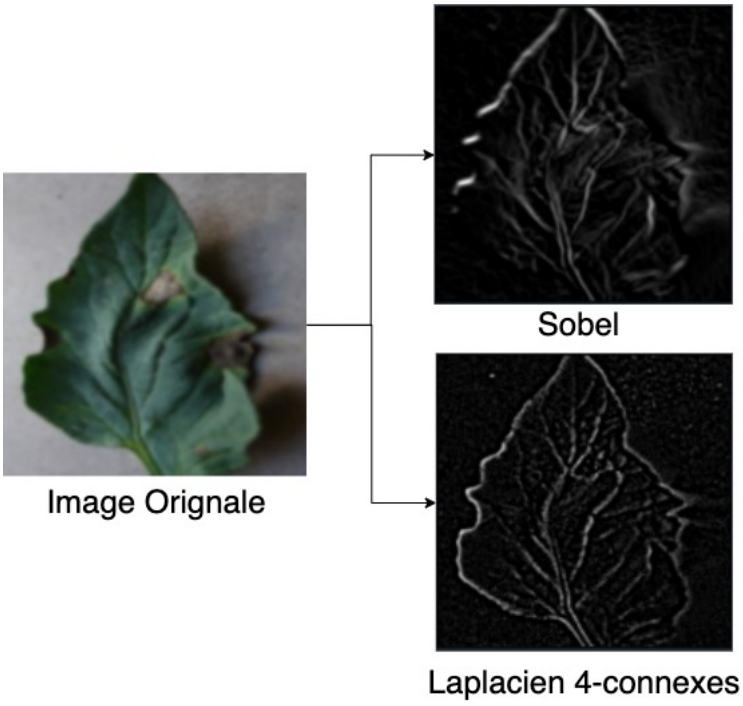


FIG. 1.7 : Détection des contours d'une image

(d) **Segmentation des images**

La segmentation est utilisée pour partitionner l'image dans le but de trouver les régions d'intérêt. Elle vise à isoler la région présentant des anomalies : Cette représentation simplifiée de l'image est facile à analyser et plus significative pour différencier les régions infectées et non infectées. La figure 1.8 montre deux exemples de segmentation d'une plante infectée.

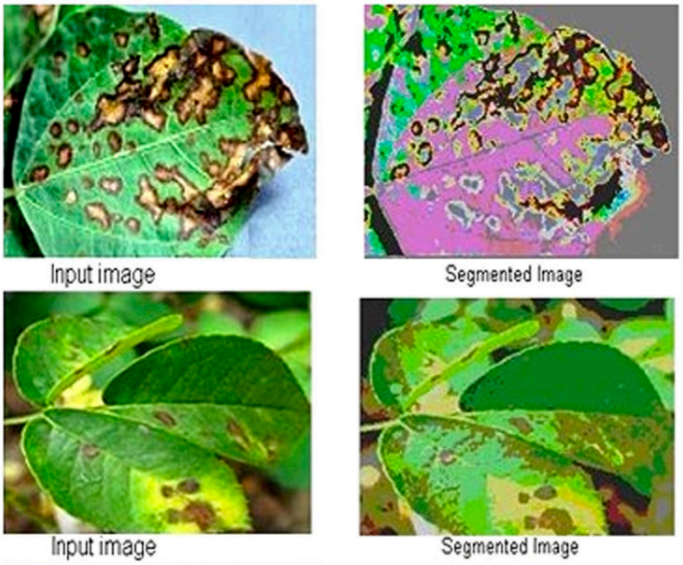


FIG. 1.8 : Exemples des images avant et après la segmentation [14]

1.4.3 Génération des caractéristiques

Les caractéristiques représentent des attributs/informations pertinentes et discriminantes propre à un objet, qui permettent de le distinguer des autres objets. L'étape d'extraction des caractéristiques est très importante dans la construction du modèle de classification/reconnaissance des maladies des plantes et vise à l'extraction d'attributs pertinents caractérisant chaque maladie [15].

En ce qui concerne les plantes, l'identification et la classification des maladies sont généralement basées sur l'analyse des feuilles. Parmi les descripteurs pouvant être utilisés pour extraire des caractéristiques, nous distinguons les descripteurs de texture et les descripteurs de gradient.

En effet, ces deux informations sont fortement corrélées car bien visibles sur les contours ou régions hétérogènes de l'image. En termes de texture, les descripteurs les plus couramment utilisés sont les matrices de co-occurrence et les motifs binaires locaux (LBP), tandis que pour les gradients, nous calculons principalement HOG (Histograms of Oriented Gradients) et SIFT (Scale Invariant Feature) [16]. Plusieurs autres descripteurs hybrides, tels que Gradient-LBP, sont utilisés dans la littérature pour améliorer conjointement les informations de texture et de gradient.

Plus précisément, dans [17], les auteurs ont utilisé plusieurs variantes de LBP pour développer un système de détection des maladies du raisin. De plus, HOG a été utilisé comme descripteur local pour identifier les maladies du soja [18]. Les résultats publiés montrent des performances satisfaisantes dans la classification de la rouille et des moisissures. De plus, afin de représenter les caractéristiques locales des images, des descripteurs SIFT qui offrent de très bonnes performances dans plusieurs domaines sont utilisés dans [19], pour détecter les infections dans les images de feuilles de soja. Dans ce travail, le système de détection est basé sur un classifieur SVM qui utilise les points clés SIFT comme attributs pour distinguer les feuilles saines des feuilles infectées. Dans un autre travail, on note l'utilisation de matrices de co-occurrences de couleurs comme descripteurs [20].

Le système proposé est conçu pour détecter les maladies des agrumes avec des caractéristiques de couleur extraites du canal HSI via une matrice de cooccurrence. En plus des descripteurs référencés dans cette section, tous les descripteurs connus en reconnaissance de formes peuvent être utilisés.

Toutefois, avec l'envahissement de ce domaine par les systèmes basés sur les réseaux de neurones convolutifs (Convolutional neural networks CNN), l'utilisation des descripteurs dissociés du modèle de classification est limitée à quelques travaux de recherche.

1.4.4 Classification et détection de Maladie

L'étape de classification est la plus importante de la détection des maladies des plantes à l'aide de la vision par ordinateur et du traitement d'image. Compte tenu de son importance dans la détection des maladies, les performances de cette phase dépendent des étapes précédentes telles que l'acquisition des données, l'étape de prétraitement, la segmentation de la zone infectée et l'extraction et la sélection finales des caractéristiques.

Dans les premiers travaux de recherche en agriculture intelligente, nous avons découvert des classifieurs classiques tels que les K-plus proches voisins et les classifieurs bayésiens. Plus récemment, les SVM et les réseaux convolutifs ont été largement utilisés. Par conséquent, les classifieurs adoptés dans l'état de l'art peuvent être résumés comme suit :

Classifieur K plus proches voisins (KPVV)

L'algorithme du k plus proche voisin est une méthode d'apprentissage basée sur les instances. Cette méthode ne nécessite pas de phase d'apprentissage, elle construit un modèle basé sur les échantillons d'apprentissage associés à la classe la plus proche voisine (vote majoritaire) avec une fonction de distance et une fonction de sélection de classe. Dans [21], les auteurs ont évalué l'utilisation de ce classifieur pour détecter les maladies fongiques dans les feuilles de maïs.

Support Vector Machines (SVM)

SVM est un modèle discriminatif qui tente de minimiser l'erreur d'apprentissage tout en maximisant la marge séparant les données de deux classes. C'était un classifieur très réussi dans les années 90 car il surmontait les limites des réseaux de neurones multicouches, à savoir le surapprentissage des données et le grand nombre de paramètres que l'utilisateur doit régler. Semblable à d'autres applications de traitement de données, SVM a été utilisé dans plusieurs travaux de recherche comme système de détection automatique des plantes malades [21], [22]. Il fournit généralement une plus grande précision que les classifieurs classiques.

Classifieurs de Bayes

Ces classifieurs utilisent le théorème de Bayes pour calculer la probabilité d'appartenir à une classe. Ces classes sont représentées par des modèles probabilistes, tels que des modèles gaussiens. Dans [22], ce classifieur a été utilisé pour détecter les maladies dans la base de données Plant Village, qui comprenait plusieurs cultures. Ce classifieur offre une précision inférieure à 85%.

Random Forest

Ces classifieurs utilisent le théorème de Bayes car il s'agit d'un ensemble d'arbres de décision qui sont des représentations visuelles d'algorithmes de classification de données selon différents critères que l'on appellera décisions (ou nœuds).
Ce classifieur est équivalent à un ensemble de règles de décision dont l'exécution route les attributs des données de test vers une classe donnée. Chaque règle de décision peut être équivalente à la fonction de décision d'un classifieur indépendant. Concernant son application dans le domaine de la détection des plantes malades, ce classifieur a été utilisé par rapport aux classifieurs KPPV et bayésiens, aux SVM et aux réseaux de neurones [22]. Pour générer des caractéristiques, nous considérons des descripteurs de couleur avec 20 paramètres de texture extraits de la matrice de cooccurrence. Les résultats obtenus montrent que SVM offre les meilleures performances.

Réseaux de neurones artificiels

C'est l'algorithme le plus utilisé dans le domaine de la classification et la reconnaissance des maladies des plantes grace à sa capacité de calcul, ce qu'a fait qu'actuellement, au moins 90% des travaux portant sur les maladies des plantes utilisent les réseaux de neurones artificiels. Les réseaux de neurones artificiels sont considérés comme des machines conçues pour modéliser la manière dont le cerveau exécute une tâche. Il est composé de plusieurs couches inter-connectées de nœuds représentant une imitation des cellules nerveuses.
Pour constituer une vue précise sur l'état de l'art, la section 1.5 rapporte les résultats de quelques travaux de recherche réalisés sur la détection et la classification des plantes malades.

1.5 Travaux connexes

Cette section réalise une analyse des dernières recherches pour l'application de l'apprentissage profond dans le domaine de la détection des maladies des plantes et plus précisément les travaux faits en utilisant l'ensemble de données **PlantVillage** [23], les résultats sont résumés dans le tableau 1.1.

les auteurs de [24] ont proposé un système de détection des maladies virales des plantes. L'ensemble de données utilisé était composé de 800 images de feuilles de concombre représentant deux maladies différentes et également des feuilles saines. Les auteurs ont utilisé

leur propre modèle CNN. Le système proposé a atteint une précision de classification maximale de 94,9% dans le cadre d'une stratégie de validation croisée à 4 volets. Les mêmes auteurs ont mené une autre recherche [25] pour détecter sept types de maladies virales du concombre. Ils ont utilisé un ensemble de données de 7250 images comprenant des maladies virales et des feuilles saines. Les classifieurs ont atteint une précision moyenne de 82,3% avec une stratégie de validation croisée quadruple.

Les auteurs de [26] ont utilisé un CNN pour reconnaître 13 différents types de maladies végétales à partir de feuilles saines. L'ensemble de données utilisé a été téléchargé sur Internet et se composait de plus de 3000 images originales, représentant 13 maladies différentes dans différentes cultures et deux autres classes pour les feuilles saines et les images d'arrière-plan. Les auteurs ont utilisé un modèle CNN CaffeNet pré-entraîné qui a atteint une précision comprise entre 91% et 98%, pour les tests de classes séparées et une précision globale de 96,3%.

Un grand ensemble de données public appelé PlantVillage [23] dont les échantillons sont présentés dans la figure 1.1 de près de 54 300 images recueillies dans des conditions contrôlées a été utilisé par [27], où les auteurs ont utilisé un CNN profond pour identifier 14 espèces de cultures et 26 maladies (ou leur absence) avec un total de 38 classes différentes. Ils ont utilisé deux modèles différents, à savoir AlexNet et GoogLeNet. La précision maximale obtenue était de 99,3% lors de l'utilisation de GoogLeNet avec des images RVB et l'apprentissage par transfert, cependant lors du test du système proposé avec des images téléchargées sur internet, la précision a chuté à seulement 31,4%.

Les auteurs de [28] ont utilisé l'architecture LeNet pour détecter deux types de maladies parmi les maladies saines dans des images de feuilles de bananiers recueillies dans le cadre du projet Plant Village. l'ensemble de données contenait 3700 images et le modèle a atteint une précision maximale de 99,72%. Une autre utilisation d'ensemble de données PlantVillage a fait appel à l'apprentissage profond pour détecter la gravité de la maladie du black rot des pommes [29]. La partie d'ensemble de données qu'ils ont utilisée contient plus de 150 images à quatre stades de gravité. Pour le modèle CNN, les auteurs ont utilisé et comparé les résultats de quatre modèles différents (VGG16, VGG19, Inception-V3 et ResNet50), qui ont été ajustés et formés à partir de zéro. Ils ont atteint une précision maximale de 90,4% avec le modèle VGG16 ajusté.

Il existe encore de nombreuses expériences et articles qui ont fait de nombreuses recherches dans la classification des maladies agricoles qu'on va tous les citer dans le tableau ci-dessous :

Auteurs	Ensemble de données			Modèle	Précision	Année
	Type de plante	Nombre de classes	Nombre d'images			
[30]	Maize	9	500	GoogLeNet, CifarlO	98.9%, 98.8%	2018
[31]	14 crop species	38	54300	VGG16, Inception V4, ResNet50 ,101,152 et Dense nets avec 121 couches	81.83%, 98.08%, 99.59%, 99.66%, 99.59%, et 99.75%	2018
[32]	Tomato	7	13262	AlexNet, VGG16. avec Transfer learning.	97.49%, 97.29%	2018
[33]	Cucumber	4	1184	Pre-entrainé AlexNet	93.2%, 94%	2018
[34]	19 crops	38	56000	Inception v3, Mobile Net	88.6%1 92%	2018
[28]	Tomato	5	500	RGB based CNN comme extracteur de caractéristiques, LVQ comme classifieur	86%	2018
[35]	Banana	3	3700	LeNet	99.72%	2017
[36]	Apple	4	2086	VGG16, VGG19, Inception-v3, et ResNet50	90.4% avec VGG 16 model.	2017
[37]	Tomato	10	18000	AlexNet, SqueezeNet	95.65%, 94.3%.	2017
[38]	Tomato	10	14828	AlexNet, GoogLeNet.	99.18% avec GoogLeNet	2017
[27]	Olive	3	299	LeNet avec transfer learning	98.6+1.47%true positive rate	2017
[25]	14 crop species	38	54300	AlexNet et GoogLeNet avec et sans transfer learning.	99.34% avec GoogLeNet et transfer learning.	2016

Tab. 1.1 : Résumé des articles sur l'utilisation de l'apprentissage profond dans la détection des maladies des plantes [39]

Tous les articles présentés ont été divisés en fonction du type d'aliment étudié ou d'ensemble de données utilisé. Les principaux aliments utilisés dans les études précédentes allaient de la tomate au riz et comprenaient également des aliments régionaux comme le manioc. L'aliment tomate est la plus abordée en raison de son importance stratégique et du grand nombre de maladies qui l'affectent. De nombreux articles ont tenté de classer un grand nombre de maladies et d'espèces, atteignant un maximum de 38 classes différentes [27] [31] [40]. l'ensemble de données le plus utilisé est l'ensemble de données PlantVillage disponible publiquement pour les maladies des plantes, soit dans son ensemble, soit en sélectionnant des aliments spécifiques à partir de celui-ci, tandis que les autres ensemble de donnéess utilisés sont des ensemble de donnéess privés qui ont été créés et utilisés par les auteurs de la recherche sans les rendre disponibles pour la communauté de recherche.

La plupart des recherches étudiées ont utilisé des modèles d'apprentissage profond pour classer les maladies des images, quelques-unes seulement ont utilisé les modèles pour segmenter les zones malades et une seule a estimé la gravité de la maladie. C'est logique, car ce qui compte vraiment, c'est de classer si la plante est malade et d'identifier cette maladie afin de déterminer le traitement approprié pour ce cas. Certains articles de recherche ont ciblé des appareils plus portables, comme les téléphones portables, et y ont déployé leur modèle. Cela est nécessaire pour l'application en temps réel des modèles, car les agriculteurs, en particulier dans les pays développés, ne disposent pas toujours de

l'équipement adéquat, de connexions internet fiables ou de suffisamment de temps pour effectuer un téléchargement d'images en plusieurs étapes vers une application fonctionnant sur un serveur Internet pour classer la maladie. Ainsi, un modèle léger avec une efficacité acceptable qui est entièrement déployé sur un mobile ou un autre dispositif dédié est considéré comme le bon choix ici.

1.6 Conclusion

Dans ce chapitre, nous avons discuté les généralités des méthodes de détection des maladies des plantes et des définitions des différentes étapes des systèmes adaptés , acquisition, prétraitement, segmentation, etc. Nous introduisons également les différents classifieurs explorés dans l'étude pour la détection des maladies des plantes. Dans le chapitre suivant, nous décrivons les méthodes utilisées pour développer notre système de détection et de classification des plantes malades.

Dans ce chapitre nous avons traité des généralités sur les méthodes de détection des maladies des plantes., ainsi que la définition du système adapté avec ses différentes étapes, Acquisition, prétraitement, segmentation et d'autres, nous avons également présenté les divers classifieurs explorés dans les études pour la détection des maladies des plantes.

La détection précoce des plantes (avant le début des symptômes de la maladie) pourrait être une source d'information pour l'exécution d'une bonne gestion de ravageuses stratégies et des mesures de lutte contre les maladies pour prévenir le développement et la propagation des maladies. Les techniques avancées de détection des maladies au sol qui pourraient éventuellement être intégrées à un véhicule agricole représentent l'un des solutions.

Aujourd'hui il est indispensable de faire la mise en œuvre des stratégies de gestion appropriées telles que les applications de fongicides, des produits chimiques spécifiques contre les maladies. Ces stratégies qu'elle nous permet l'application de pesticides grâce à des informations instantanées sur la santé des plantes. Cela facilite le contrôle des maladies et l'amélioration de la productivité.

Chapitre 2

Réseaux de neurones convolutifs

2.1 Introduction

L'Intelligence Artificielle (IA) a fait l'objet d'une médiatisation et d'une attention sans précédent ces dernières années, générant de nombreuses promesses mais aussi des craintes, certaines fondées sur des visions très spéculatives ou très lointaines des capacités des machines. Ce fort intérêt pour l'intelligence artificielle est particulièrement pertinent pour les avancées technologiques majeures qui ont permis d'augmenter significativement les performances des ordinateurs dans de nombreux domaines, comme la reconnaissance automatique de la parole ou la vision par ordinateur [41]. Ces avancées ouvrent la perspective d'introduire l'intelligence artificielle sous différentes formes. dans l'environnement de travail. On notera en particulier le nombre croissant d'industries qui font l'objet d'une attention particulière (santé, agriculture, finance, banque, assurance, transport, etc.).

Ce chapitre présente un aperçu théorique de l'évolution des réseaux de neurones artificiels vers les réseaux de neurones convolutifs utilisé dans ce travail pour le diagnostic des maladies des plantes.

2.2 Intelligence artificielle

Les progrès récents de l'intelligence artificielle et de l'apprentissage automatique ont changé la façon dont nous traitons, analysons et manipulons les images. Ceci est largement dû à la vague d'enthousiasme pour l'apprentissage automatique profond, en tant que nouvelle frontière de l'intelligence artificielle, où les caractéristiques les plus représentatives et discriminantes sont apprises de bout en bout. Plus précisément, les réseaux de neurones convolutifs (Convolutional Neural Networks : CNN) sont les meilleurs parmi les algorithmes d'apprentissage pour comprendre le contenu des images, et ont montré des résultats exemplaires dans les tâches liées à la segmentation, la classification, la détection, et l'identification [42] [43]. En règle générale, un CNN se compose de deux blocs de traitement, un bloc convolutif qui génère des caractéristiques et un bloc de prédiction qui effectue l'étape de classification (ou de détection). Pour cette raison, il existe deux modes d'utilisation, les CNN sont soit utilisés comme un système de bout en bout intégrant la génération et la classification des caractéristiques, soit comme un bloc responsable de l'extraction des caractéristiques [44].

C'est pourquoi, dans le présent travail, nous proposons d'étudier l'utilisation des CNN comme système de bout en bout et comme extracteur de caractéristiques. Pour l'implémentation de ce système, nous adoptons plusieurs architectures comme VGG-16, AlexNet, ResNet qui sont les plus utilisées dans les applications de vision par ordinateur ainsi qu'un modèle qu'on va développer de zéro.

2.3 Notions sur les réseaux de neurones artificiels

L'intérêt pour les réseaux neuronaux a commencé avec les études élaborées pour comprendre le cerveau, la cognition et la perception de l'humain. . En fait, les réseaux neuronaux artificiels sont nés de la fascination pour les capacités humaines d'apprentissage, de compréhension et d'exécution de tâches complexes avec simplicité et élégance précise [45]. À l'heure actuelle, divers modèles de réseaux neuronaux artificiels sont capables d'exécuter efficacement les mêmes tâches que les humains, mais avec plus de précision, plus rapidement et sans jamais se fatiguer ni se soucier de travailler dans des conditions insoutenables.

2.3.1 Du neurone biologique au neurone artificiel

Le cerveau humain est constitué d'un très grand nombre de cellules nerveuses (neurones) qui sont reliées entre elles par des synapses. Le nombre de neurones, déterminé à la naissance, est estimé à 10 milliards et reste relativement constant tout au long de la vie. Comme le montre la figure 2.1, un neurone possède une sortie unique appelée axone et plusieurs entrées appelées dendrites. Les axones peuvent être ramifiés pour se connecter à plusieurs autres neurones via leurs dendrites. Des signaux, appelés potentiels d'action, sont envoyés entre les neurones. Plus précisément, les signaux sont reçus via les dendrites et additionnés à l'intérieur d'un noyau appelé soma. Ensuite, un seuil doit être atteint pour que le signal accumulé soit propagé le long de l'axone vers d'autres neurones. Lorsqu'un être humain apprend quelque chose de nouveau, le processus d'impression de la mémoire dans le cerveau a lieu. Les synapses entre les neurones stimulés sont renforcées au moment même où l'apprentissage se produit. Ainsi, c'est à la fois dans les synapses et dans les neurones avec leur seuil, que toute la mémoire est stockée.

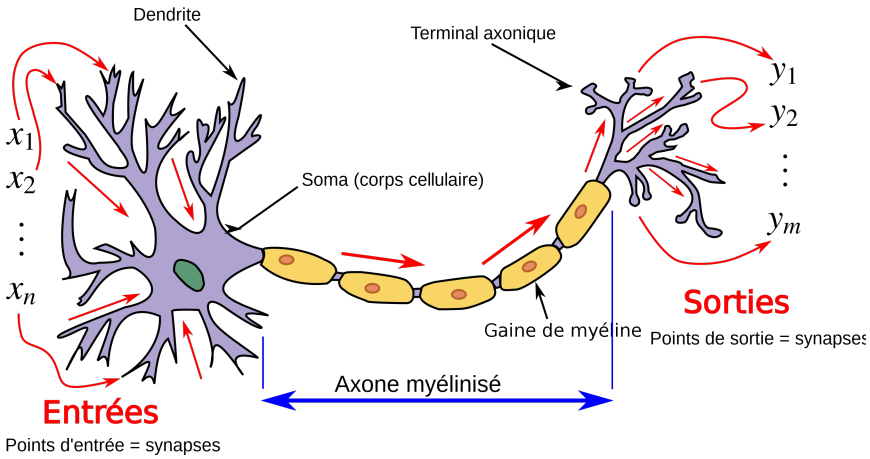


FIG. 2.1 : Neurone biologique [46]

2.3.2 Perceptron

Les réseaux de neurones artificiels (Artificial Neural Networks : ANN) visent à reproduire le processus d'apprentissage du cerveau humain. Un ANN est basé sur un rassemblement de nœuds connectés appelés neurones artificiels, qui tentent de modéliser les neurones d'un cerveau biologique.
Le réseau neuronal existant le plus basique est le perceptron, qui est composé d'un seul neurone formel. Il est considéré comme le processus d'apprentissage le plus ancien qui ait été réalisé, pour la classification de deux classes linéairement séparables. Un perceptron est composé d'une ou plusieurs entrées, d'une unité de traitement et d'une sortie.
La figure 2.2 montre la structure d'un neurone artificiel qui procède par la sommation pondérée de son vecteur d'entrée :

$$A = (a_1; a_2; \ldots; a_M) \in \Re^M \tag{2.1}$$

Et le vecteur de poids synaptique :

$$W = (w_1; w_2; \ldots; w_M) \in \Re^M \tag{2.2}$$

La somme obtenue dite potentiel est évaluée par une fonction d'activation qui fournit une sortie y. La fonction d'activation imite le fonctionnement du soma (noyau). Elle effectue la somme des entrées pondérées et la compare à un seuil pour activer ou non la sortie en envoyant un potentiel somatique.

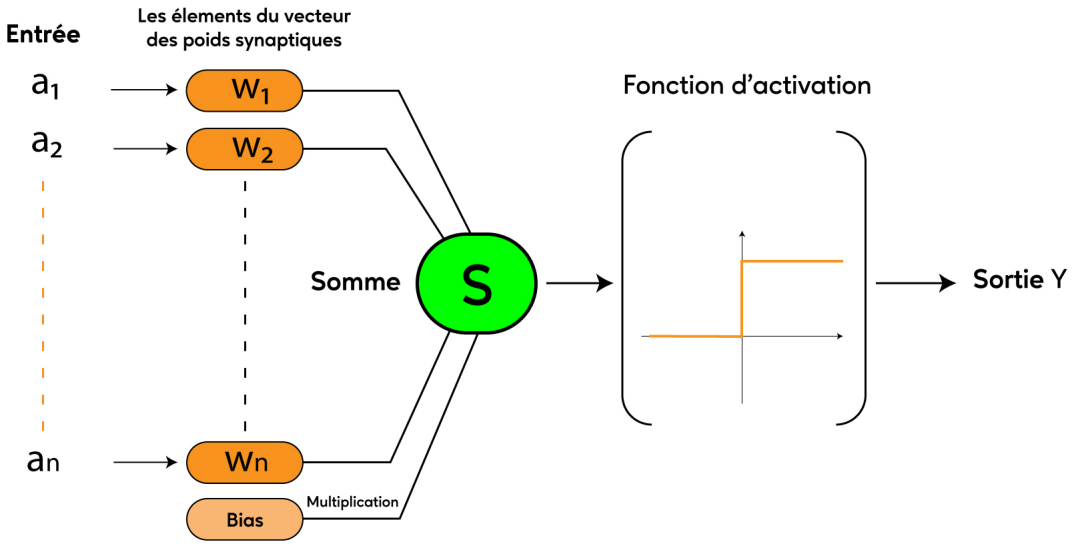


FIG. 2.2 : Structure d'un neurone artificiel [47]

2.3.3 Le perceptron multicouche

Le Perceptron multicouche (PMC) a été introduit en 1986 par Rumelhart et al., [48] comme une extension du perceptron monocouche précédent. Il se compose de trois types

de couches : Chaque neurone est lié à la totalité des neurones des couches adjacentes, ce qui donne un réseau entièrement connecté, comme illustré par la figure 2.3. La dimension des couches d'entrée et de sortie détermine la nature du problème à résoudre. En d'autres termes, le nombre de neurones dans la couche d'entrée détermine la dimension de la donnée traitée, tandis que le nombre de neurones dans la couche de sortie détermine le nombre de classes. Pour ce qui est des couches cachées, leurs nombres et le nombre de neurones qui les constituent est un problème de conception. Avec peu de neurones cachés, le modèle ne sera pas en mesure d'apprendre des limites de décision complexes. Au contraire, trop de neurones réduisent la généralisation du modèle dû à un sur-apprentissage. Ainsi, un réglage expérimental est nécessaire pour trouver le meilleur compromis entre le nombre de nœuds cachés et la performance de généralisation du réseau.

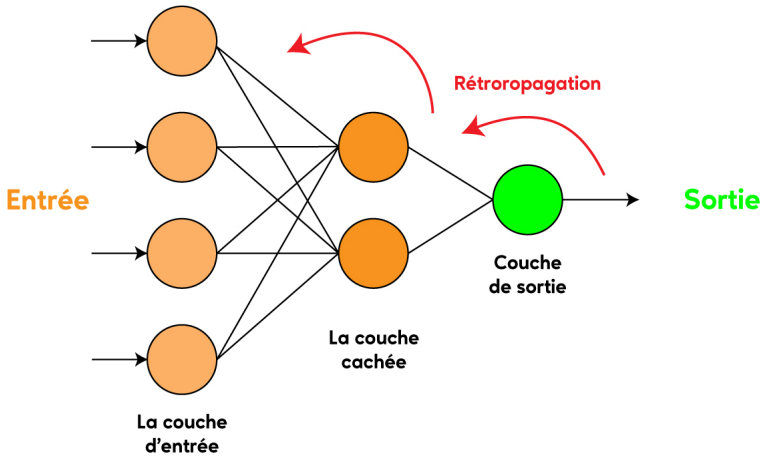


FIG. 2.3 : Représentation schématique d'un MLP avec une seule couche cachée

Le PMC est entrainé par l'algorithme de retropropagation du gradient de l'erreur proposé par Rumelhart et al., [48]. Il permet aux informations de circuler en sens inverse dans le réseau afin de calculer le gradient. Il consiste à ajuster les poids synaptiques de tous les neurones des différentes couches en calculant l'erreur quadratique moyenne E entre le vecteur de sortie estimé y et le vecteur de sortie réelle y_r par l'équation suivante :

$$E = \frac{1}{2}\sum_{i=1}^{N}\left(y_r^i - y^i\right)^2 \tag{2.3}$$

Les poids synaptiques sont ensuite modifiés tel que :

$$\begin{aligned} w(t+1) &= w(t) + \Delta w(t+1) \\ \Delta w(t+1) &= -\alpha\frac{\partial E}{\partial w} + \mu\Delta w(t) \end{aligned} \tag{2.4}$$

t : itération en cours (elle correspond au passage d'une donnée à travers le réseau).
$\Delta w()$: Changement de poids au fil des itérations.
$\frac{\partial E}{\partial w}$: gradient de l'erreur par rapport au poids w.

Il est à noter que l'objectif est de minimiser une fonction cout représentée par l'erreur quadratique moyenne E jusqu'à son minimum local ce qui est appelée descente du gradient.

Quant au paramètre α, il représente le taux d'apprentissage ou le pas d'apprentissage, qui constitue le pas que fait la descente du gradient dans la direction du minimum local déterminant ainsi la rapidité ou la lenteur avec laquelle le minimum local et donc les poids optimaux sont approchés. La figure 2.4 montre l'impact et l'importance du pas d'apprentissage.

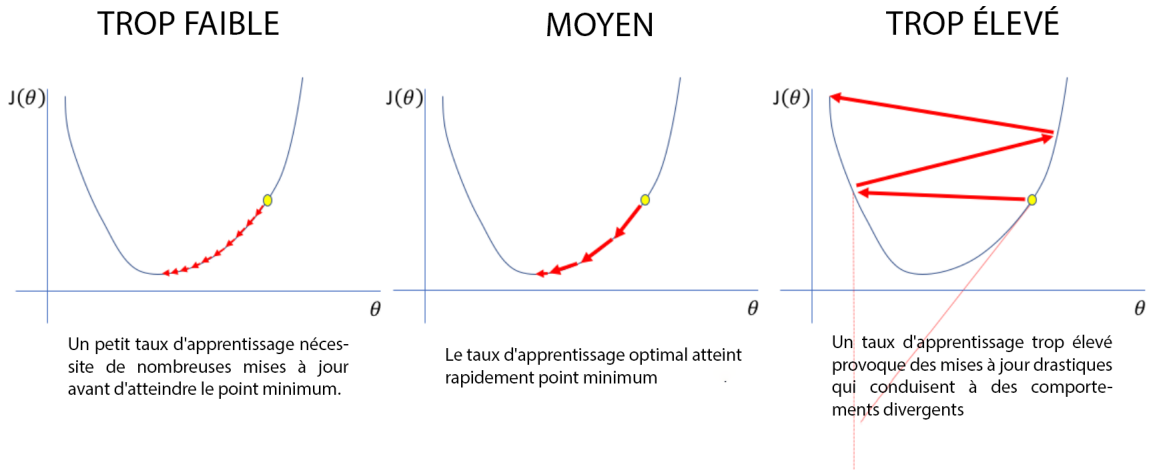


Fig. 2.4 : Illustration de l'architecture CNN

D'un autre coté le paramètre μ appelé momentum permet de conserver les informations relatives à la dernière mise à jour des poids synaptiques (itération t) pour en tenir compte dans la mise à jour actuelle (itération $t+1$). Il évite ainsi de rester coincé dans un minimum local ainsi que les effets d'oscillations.

2.4 Réseaux de neurones convolutifs CNN

Les réseaux de neurones convolutifs (CNN ou ConvNet) sont une extension du PMC. Ils sont conçus pour extraire automatiquement les caractéristiques des données d'entrée. Ils ont été initialement inspirés par la découverte faite par Hubel et Wiesel, en 1962, où ils ont noté que des modèles particuliers stimulaient l'activité dans des parties spécifiques du cerveau du chat [49]. La première utilisation de ces réseaux a été réalisée par Fukushima avec la carte auto-organisatrice (SOM) pour l'extraction de caractéristiques par apprentissage non supervisé [50]. En reconnaissance de formes, un développement important a été atteint par LeCun et al., dans [51], qui ont proposé l'architecture LeNet pour la reconnaissance des chiffres manuscrits.

CNN élimine le besoin de l'extraction manuelle des caractéristiques. En effet, il extrait les caractéristiques et attribue des poids et des biais apprenables à divers aspects de l'image afin de les différencier les unes des autres. Techniquement, chaque image d'entrée passera par deux blocs ; bloc de convolution et bloc de classification.

De manière générale, le bloc de convolution peut contenir trois types de couches à savoir, les couches de convolution contenant les différents filtres, les couches de sous-échantillonnage (pooling) et une couche d'aplatissement pour assurer la liaison avec le bloc de classification. Ce dernier n'est qu'un perceptron multicouches pouvant contenir une ou plusieurs couches intermédiaires (Fully connected layers), et une couche de sortie. La figure 2.5 illustre l'architecture d'un CNN de détection des plantes malades.

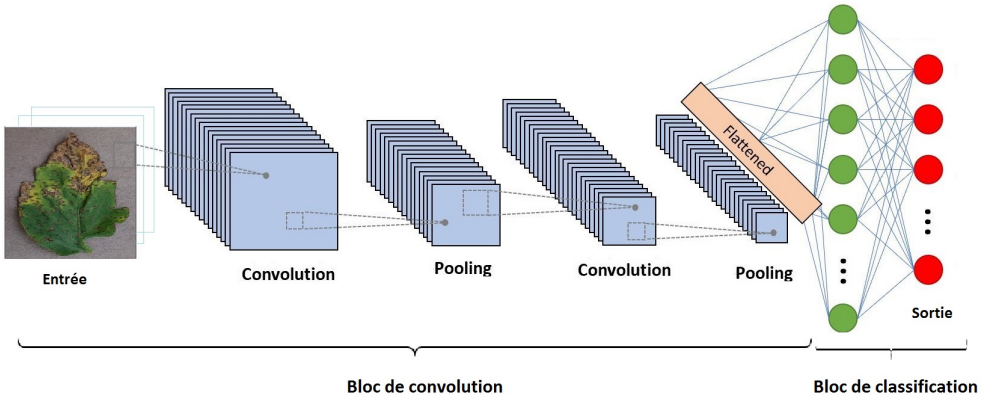


FIG. 2.5 : Illustration de l'architecture CNN

2.4.1 Couche de convolution (Convolutional Layer)

Une couche convolutive 2D extrait les caractéristiques d'une image d'entrée et préserve la relation entre les pixels en apprenant ses caractéristiques à l'aide de filtres carrés. L'opération de convolution prend deux entrées : une matrice d'image de dimension $hxlxd$ (où h est la hauteur, w est la largeur et d est le nombre de canaux) et un filtre ou un noyau de dimension d. Le nombre de filtres utilisés détermine la profondeur du traitement. Typiquement, l'entrée de la couche convolutive est une image couleur RVB, elle est donc représentée par 3 canaux. Dans le cas d'une image binaire ou en niveaux de gris, l'image d'entrée de la couche convolutive est représentée par un seul canal. La matrice de sortie est calculée avec un produit scalaire séquentiel entre le filtre et la matrice d'image [52]. Le filtre est déplacé sur l'image de gauche à droite, de haut en bas, avec un certain nombre de pas appelées strides, pour effectuer le même calcul (voir figure 2.6).

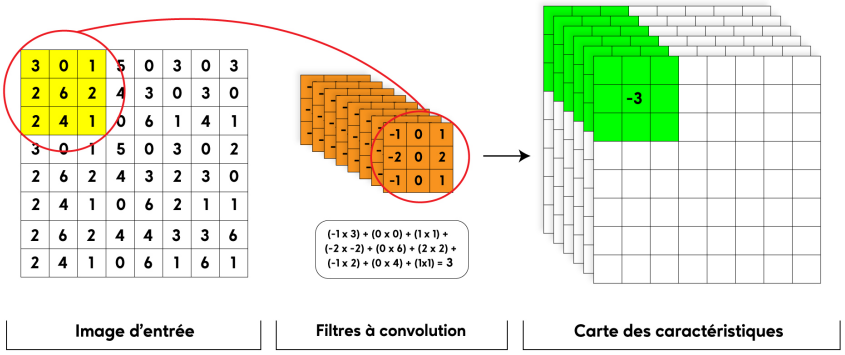


FIG. 2.6 : Exemple d'opération de convolution.

2.4.2 Couche de Pooling (Pooling layer)

La couche de pooling intervient à l'issue de chaque couche de convolution. Elle effectue une opération de sous-échantillonnage en fonction des dimensions spatiales largeur et hauteur uniquement. L'image d'entrée est divisée en une série de carrés non superposés. Ce type de couche sert à réduire le nombre de paramètres et la puissance de calcul requise, tout en préservant les caractéristiques les plus importantes. En pratique, l'image est divisée en petites cellules carrées adjacentes, espacées les unes des autres d'un stride, afin de ne pas perdre trop d'informations. Le même nombre d'images de sortie que d'images d'entrée est obtenu mais chaque image a un nombre de pixels inférieur. Il existe deux principaux types pooling : max-pooling and average pooling. Max-pooling renvoie la valeur maximale de la partie de l'image couverte par le noyau. Tandis que, average pooling renvoie la moyenne de toutes les valeurs de la partie de l'image couverte par le noyau. En pratique, max-pooling est la plus utilisée et fonctionne bien mieux que average pooling. La figure 2.7 représente un exemple d'opérations de max-pooling and average pooling.

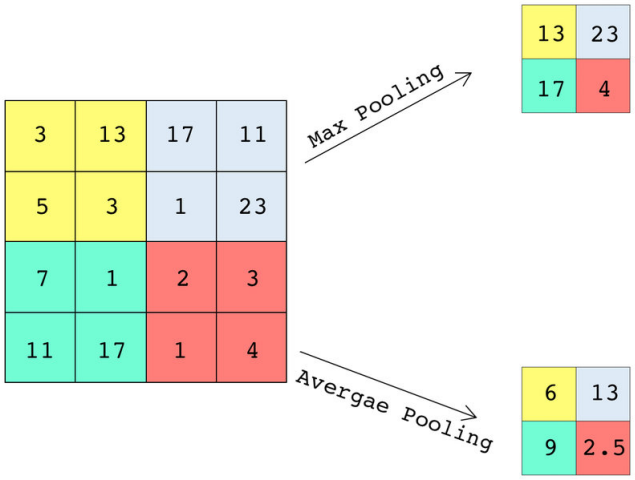


FIG. 2.7 : Exemple d'opération de pooling.

2.4.3 Couche d'Aplatissement (Flatten Layer)

Le bloc convolutif est terminé par une couche d'aplatissement qui assure la connexion avec le bloc de prédiction qui est composé de l'ensemble des cartes caractéristiques dont le nombre correspond au nombre de filtres utilisés dans la dernière couche de convolution en un vecteur unidimensionnel qui est ensuite transmise au réseau neuronal pour traitement. La figure 2.8 représente un exemple du processus d'aplatissement.

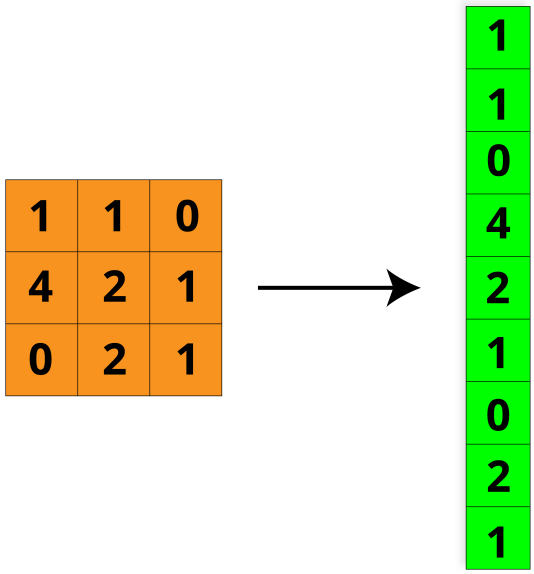


FIG. 2.8 : Mécanisme d'aplatissement

2.4.4 Paramètres d'un CNN

Filtre

La convolution consiste à faire glisser une série de filtres ou noyaux de convolution sur l'image d'entrée. Le choix des filtres, de leur taille et de leur nombre (appelé aussi profondeur) est une tâche expérimentale. La "profondeur de la couche à convolution" fait référence au nombre de noyaux utilisés, ce qui est à l'origine du nom "Deep learning". Si nous utilisons un nombre q de filtres, l'image sera convoluée g fois. Ce processus crée 9 matrices filtrées appelées "cartes de caractéristiques". Un exemple de filtrage est illustré à la figure 2.9. Dans la plupart des architectures CNN prédéfinies, il existe toujours un choix par défaut spécifique à l'application pour laquelle le modèle a été construit [51].

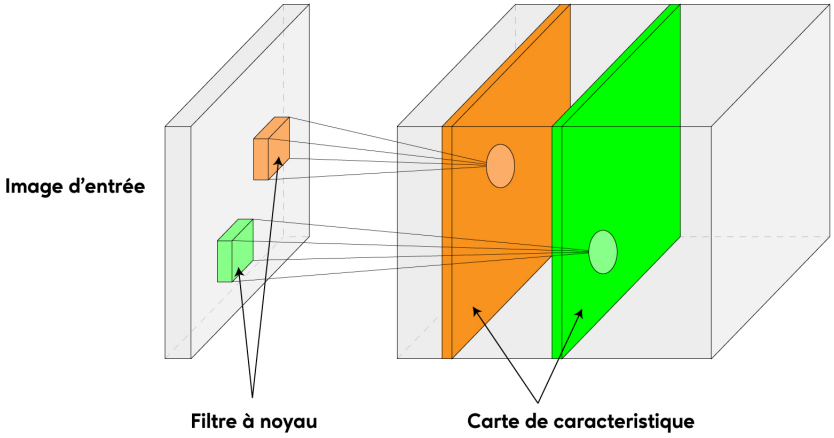


FIG. 2.9 : Cartes de caractéristiques

Stride

Stride est le nombre de pixels décalés sur la matrice d'entrée. Lorsque le stride est de 1, nous déplaçons les filtres d'un pixel à la fois. Lorsque le stride est de 2, nous déplaçons les filtres de 2 pixels à la fois, et ainsi de suite.

Zéro Padding

Après chaque couche convolutive, la taille de l'image résultante est plus petite que celle de l'image d'entrée, car le filtre ne peut pas traiter les pixels de bords. Si plusieurs couches convolutionnelles sont utilisées, l'image de sortie sera très petite par rapport à l'image d'entrée. Pour conserver la même taille que l'image d'entrée, le zéro padding ajoute des zéros aux contours de l'image, en tenant compte de la taille du filtre. De cette façon, quel que soit le nombre de couches convolutionnelles, l'image de sortie a la même taille que l'image d'entrée.

Les fonctions d'activation

La fonction d'activation est une fonction mathématique appliquée sur la somme pondérée des entrées. Elle va reproduire le potentiel d'activation que l'on retrouve dans le neurone biologique qui permet de transmettre l'information aux autres neurones si le seuil de stimulation est atteint. Aujourd'hui, plusieurs fonctions d'activation ont été proposées et nous citons les plus utilisées :

- **Sigmoid**
  La fonction sigmoide est la fonction la plus ancienne et la plus utilisée. Elle représente la fonction de répartition de la loi logistique. Elle est souvent utilisée dans les réseaux de neurones parce qu'elle est dérivable, ce qui est une contrainte pour l'algorithme de rétropropagation lors du calcul du gradient de l'erreur. Pour une

entrée z, la sortie sigmoïde est donnée par :

$$F(z) = \frac{1}{1 + \exp(-z)} \tag{2.5}$$

- **Softmax**
  Couramment utilisée dans la dernière couche du réseau, c'est une fonction de normalisation qui donne une approximation de la probabilité qu'une classe soit correcte. La valeur de probabilité est calculée comme suit :

$$F(z) = \frac{\exp(z_i)}{\sum_j \exp(z_i)} \tag{2.6}$$

- **Rectified Linear Unit (ReLU)**
  Principalement appliqué dans les réseaux de neurones convolutifs, Elle permet de mettre à 0 toutes les valeurs négatives et de conserver les valeurs positives inchangées. Elle est utilisée afin d'augmenter la non-linéarité du réseau et est décrite par l'équation suivante :

$$F(z) = \max\{0, z\} \tag{2.7}$$

Batch normalization

La normalisation par lots (Batch Normalization) est une méthode algorithmique qui rend l'apprentissage des réseaux de neurones profonds (DNN) plus rapide et plus stable. Elle consiste à normaliser les vecteurs d'activation des couches cachées en utilisant le premier et le second moment statistique (moyenne et variance) du lot courant. Cette étape de normalisation est appliquée juste avant (ou juste après) la fonction non linéaire.

Dropout

Il s'agit d'une technique utilisée lors de la phase d'apprentissage, conçue pour réduire l'erreur généralisation et éviter le problème de sur-apprentissage lié aux couches entièrement connectées. Le principe d'utilisation du dropout est d'éteindre aléatoirement des neurones pour que le réseau soit plus réactif et donc puisse apprendre plus vite. La figure 2.17 présente un exemple de la technique [53].

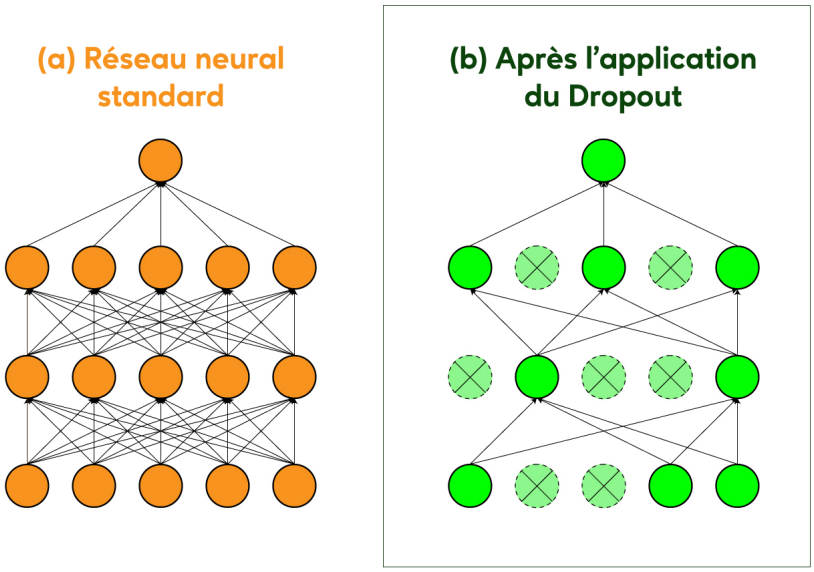


FIG. 2.10 : Mécanisme de dropout dans un réseau neuronal multicouche.

2.5 Architectures utilisées

Dans notre travail, on s'intéresse à l'utilisation des modèles (ResNet152, Vgg16, AlexNet) qui constituent des architectures de référence des CNN, pour la classification et la détection des maladies agricoles.

2.5.1 Modèle 01: AlexNet

AlexNet a été conçu par Hinton, lauréat du concours ImageNet 2012, et son étudiant Alex Krizhevsky. C'est également après cette année que des réseaux neuronaux plus nombreux et plus profonds ont été proposés. Il a atteint un erreur de 15.3% en TOP-5, ce qui était déjà remarquable par rapport aux algorithmes de classification traditionnels d'apprentissage automatique.

Architecture

L'architecture du modèle AlexNet se compose de huit couches : cinq couches convolutionnelles et trois couches entièrement connectées comme le montre la figure 2.11. Selon [54] ce n'est pas ce qui fait la particularité d'AlexNet ; voici quelques-unes des caractéristiques utilisées qui constituent de nouvelles approches des réseaux de neurones convolutifs :

- **Non-linéarité ReLU** : AlexNet utilise des unités linéaires rectifiées (ReLU) au lieu de la fonction tangente hyperbolique (tanh), qui était standard à l'époque. L'avantage de ReLU réside dans le temps d'entraînement ; un CNN utilisant ReLU a pu atteindre une erreur de 25% sur le ensemble de données CIFAR-10 six fois plus rapidement qu'un CNN utilisant tanh.

- **GPU multiples** :
  À l'époque, les GPU disposaient encore de 3 gigaoctets de mémoire. C'était d'autant plus difficile que l'ensemble d'entraînement comportait 1,2 million d'images. AlexNet permet l'entraînement multi-GPU en plaçant la moitié des neurones du modèle sur un GPU et l'autre moitié sur un autre GPU. Cela permet non seulement d'entraîner un modèle plus grand, mais aussi de réduire le temps d'entraînement. Notons, que AlexNet possède 62.3 millions de paramètres.

- **Pooling en chevauchement** (Overlapping Pooling) :
  Traditionnellement, les CNN "poolent" les sorties de groupes de neurones voisins sans chevauchement. Toutefois, lorsque les auteurs ont introduit le chevauchement, ils ont constaté une réduction de l'erreur d'environ 0,5% et ont constaté que les modèles avec pooling de chevauchement ont généralement plus robustes contre le sous-apprentissage.

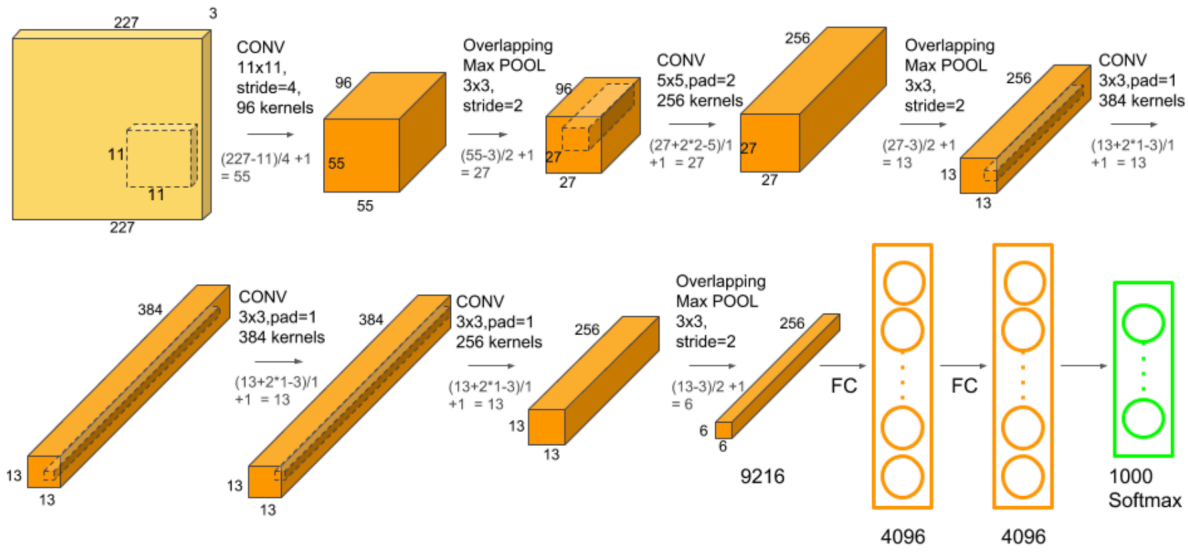


Fig. 2.11 : Architecture du modèle AlexNet

L'AlexNet a révolutionné l'implémentation des ConvNets qui se poursuit aujourd'hui, comme le ReLU et le dropout car il n'est pas facile d'obtenir de faibles erreurs de classification sans tomber de le sur-apprentissage.

2.5.2 Modèle 02: VGG16

VGG pour Visual Geometry Group est un réseau neuronal convolutif proposé par K. Simonyan et A. Zisserman de l'Université d'Oxford et a acquis une notoriété pour avoir remporté le concours ILSVRC (ImageNet Large Scale Visual Recognition Challenge) en 2014. Le modèle atteint une précision de 92,7% sur Imagenet, ce qui est l'un des meilleurs résultats obtenus. Il est à noter que ImageNet une énorme base de données de plus de 14 millions d'images étiquetées réparties dans plus de 1000 catégories. Par rapport aux

modèles précédents, il marque une amélioration en proposant un noyau de plus petite taille (3×3) dans les couches convolutives. Le modèle a été entraîné pendant des semaines à l'aide de cartes graphiques à la pointe de la technologie[55]

L'architecture

Dans la litérature, il existe deux algorithmes de VGG : VGG16 et VGG19. Dans ce travail , nous nous intéressons à l'architecture VGG16 uniquement. Si les deux architectures sont très proches et suivent la même logique, VGG19 possède plus de couches convolutives.

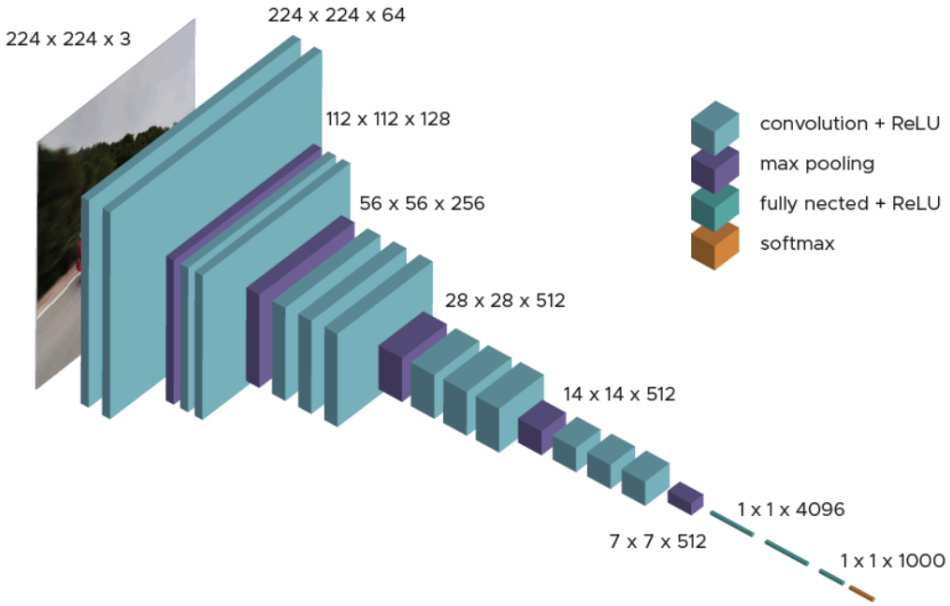


Fig. 2.12 : Architecture du model VGG16

Ce modèle améliore AlexNet en remplaçant les filtres de grande taille par une pile de filtres de taille 3x3. L'architecture de VGG-16 est illustrée à la figure 2.12.
L'entrée de la première couche convolutive est une image RVB de taille 224×224×3. L'image est passée à travers une pile de couches convolutives, où les filtres sont de taille 3×3 (qui est la plus petite taille pour capturer la notion de gauche/ droite, haut/bas, centre). La convolution a été réalisée par 13 couches convolutionnelles. Le stride de convolution est fixée à 1 pixel ; le padding spatial de l'entrée de la couche convolutive est tel que la résolution spatiale est conservée après convolution, c'est-à-dire que le padding est de 1 pixel pour les couches convolutives 3×3. Le pooling est assuré par cinq couches de max-pooling, qui suivent certaines des couches de convolution (elles ne sont pas toutes suivies par le max-pooling). Le max-pooling est effectuée sur une fenêtre de 2×2 pixels, avec un stride de 2.
Après la pile de couches de convolution et de max-pooling, nous obtenons $(7, 7, 512)$ cartes de caractéristiques, qui sont aplaties en un vecteur de caractéristiques de $(1, 25088)$. Viennent ensuite 3 couches entièrement connectées (FC), les deux premières ayant 4096 canaux chacune et la troisième effectuant une classification de 1000 classes, et contenant

ainsi 1000 canaux (un pour chaque classe). La dernière couche est la couche softmax. Notez que le nom VGG-16 provient des 13 couches convolutionnelles plus les 3 couches FC. VGG-16 se compose de 138 millions de paramètres, ce qui peut être difficile à gérer car cela nécessiterait d'énormes ressources de puissance de calcul et beaucoup de temps de formation.
Par conséquent, une approche plus pratique est utilisée, à savoir "l'apprentissage par transfert". Ce dernier est une technique d'apprentissage automatique, où un modèle entraîné pour une tâche est réutilisé comme point de départ pour entraîner un modèle sur une deuxième tâche. Dans notre cas, nous avons utilisé les paramètres du modèle VGG-16 formés sur ImageNet, comme point de départ pour la formation de notre modèle pour effectuer l'écriture.
VGG-16 a été l'une des architectures les plus performantes du défi ILSVRC 2014. Il s'est classé deuxième dans la tâche de classification avec une erreur de classification de 7,32% (juste derrière GoogLeNet avec une erreur de classification de 6,66%). Il a également remporté la tâche de localisation avec une erreur de localisation de 25,32%.
Cependant, VGG 16 contient 138 millions de paramètres, son apprentissage est donc très lent (le modèle VGG original a été entraîné sur un GPU Nvidia Titan pendant 2 à 3 semaines). De plus, la taille des poids d'ImageNet entraînés par VGG-16 est de 528 Mo. Il nécessite donc beaucoup d'espace disque et de bande passante, ce qui rend son utilisation contraignante.

2.5.3 Modèle 03: ResNet152v2

Les réseaux neuronaux convolutifs deviennent de plus en plus profonds et complexes. Il a été prouvé que l'ajout de couches supplémentaires à un réseau neuronal peut le rendre plus robuste pour les tâches liées à l'image. Mais cela peut aussi leur faire perdre en précision. C'est là que les réseaux résiduels entrent en jeu en 2015.
La tendance des spécialistes de l'apprentissage profond à ajouter autant de couches vise à extraire des caractéristiques importantes d'images complexes. Ainsi, les premières couches peuvent détecter des contours, et les couches suivantes à la fin peuvent détecter des formes reconnaissables, comme les pneus d'une voiture. Mais si nous ajoutons plus de 30 couches au réseau, ses performances en pâtissent et il fournit une faible précision. Cela va à l'encontre de l'idée selon laquelle l'ajout de couches améliore un réseau neuronal. Ce n'est pas dû à un sur apprentissage, car dans ce cas, on peut utiliser des techniques de dropout et de régularisation pour résoudre le problème[56]. Il est surtout présent à cause du populaire problème du gradient évanescent.

Le modèle ResNet152 avec 152 couches a remporté le test ILSVRC Imagenet 2015 tout en ayant moins de paramètres que le réseau VGG19, très populaire à l'époque. Un réseau résiduel est constitué d'unités ou de blocs résiduels qui ont des connexions par saut, également appelées connexions d'identité.

La sortie de la couche précédente est ajoutée à la sortie de la couche suivante dans le bloc résiduel. Le saut peut être de 1, 2 ou même 3 couches. Lors de l'ajout, les dimensions de x peuvent être différentes de $F(x)$ en raison du processus de convolution, ce qui en-

traîne une réduction de ses dimensions. Ainsi, nous ajoutons une couche de convolution supplémentaire 1×1 pour modifier les dimensions de x.
Un bloc résiduel comme le montre la figure 2.16 comporte une couche de convolution 3×3 suivie d'une couche de normalisation par lots et d'une fonction d'activation ReLU. Cette dernière est à nouveau poursuivie par une couche de convolution 3×3 et une couche de normalisation par lots. La connexion de saut saute essentiellement ces deux couches et s'ajoute directement avant la fonction d'activation ReLU. De tels blocs résiduels sont répétés pour former un réseau résiduel.

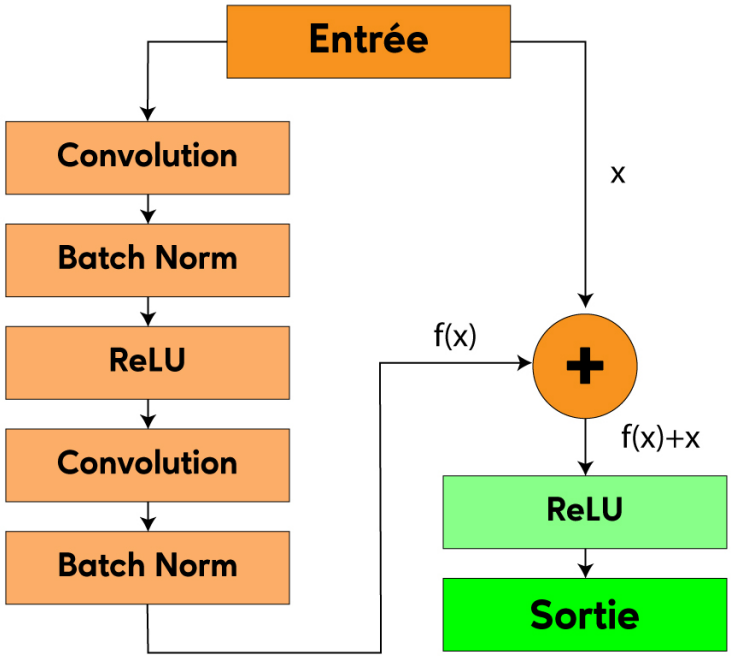


Fig. 2.13 : Un bloc résiduel

Après une comparaison approfondie de toutes les architectures CNN actuelles, le ResNet s'est distingué en détenant le taux d'erreur le plus bas du top 5%, soit 3,57%, pour les tâches de classification, dépassant toutes les autres architectures. Même les humains n'ont pas des taux d'erreur beaucoup plus bas [56].
Le tableau suivant nous montre la composition des architectures de bases du modèle ResNet à savoir : ResNet18, ResNet34, ResNet50, ResNet101 et ResNet152. :

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3x3 max pool, stride 2				
		$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix} \times 2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix} \times 3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix} \times 3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix} \times 3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix} \times 3$
conv3_x	28x28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix} \times 2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix} \times 4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix} \times 4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix} \times 4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix} \times 8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix} \times 2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix} \times 6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix} \times 6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix} \times 23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix} \times 36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix} \times 2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix} \times 3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix} \times 3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix} \times 3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix} \times 3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

Tab. 2.1 : Architectures de base du modèle ResNet

Pour conclure, on peut dire que les réseaux résiduels sont devenus assez populaires pour les tâches de reconnaissance et de classification d'images en raison de leur capacité à résoudre les gradients évanescents lors de l'ajout de couches supplémentaires à un réseau neuronal déjà profond avec un nombre de paramètres totale environ de 60 millions. Un ResNet à mille couches n'a pas beaucoup d'utilité pratique pour le moment.

2.5.4 Modèle 04: Modèle proposé

Utiliser une architecture CNN pré-entrainée telles que AlexNet, VGG16 ou encore ResNet152 demande beaucoup de ressources matérielles et de puissance de calcul en raison du nombre important de paramètres entrainables que contient chaque architecture à savoir ; plus de 62 millions pour AlexNet, plus de 138 millions pour VGG16 et plus de 140 millions pour ResNet152. D'autre part, plus le réseau est profond plus le temps d'apprentissage est important, mais aussi le temps d'inférence augmente puisqu'une donnée doit passer par toutes les couches. Par conséquent, nous proposons dans ce travail de développer notre architecture CNN de zéro pour le diagnostic des maladies des plantes et qui est beaucoup moins contraignante que les architectures pré-entrainées (voir figure 2.14). En effet, cette architecture prend comme entrée une image couleur RVB de taille 256x256 pixels qui passera par une couche de convolution avec 32 filtres de taille 3x3 pixels suivie de 5 couches de convolutions de 64 filtres de taille 3x3 pixels. Chaque couche de convolution est succédée par une fonction d'activation type ReLu et une couche max-pooling de taille 2x2 pixels avec un stride de 2 pixels. La dernière couche pooling génère 64 cartes caractéristiques de tailles 2x2 qui seront aplatit en un vecteur d'aplatissement ou flatten de 256 éléments, qui a son tour représente l'entrée de la couche cachée « Dense » ayant 64 neurones. Cette couche cachée est aussi suivie d'une fonction d'activation ReLu. La dernière couche dense représente le nombre de classe et fournit une probabilité d'appartenance à l'une des classes grâce à la fonction d'activation softmax. Cette architecture possède 184202 paramètres et est résumée dans le tableau 2.2.

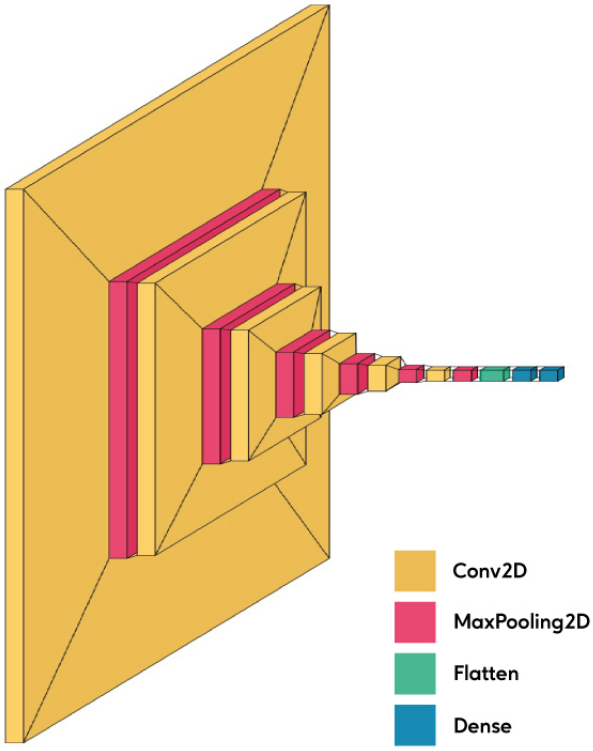


FIG. 2.14 : Architecture du modèle proposé

Type	Forme	Paramètres
Conv2D	(None, 254, 254, 32)	896
MaxPooling2D	(None, 127, 127, 32)	0
Conv2D	(None, 125, 125, 64)	18496
MaxPooling2D	(None, 62, 62, 64)	0
Conv2D	(None, 60, 60, 64)	36928
MaxPooling 2D	(None, 30, 30, 64)	0
Conv2D	(None, 28, 28, 64)	36928
MaxPooling2 D	(None, 14, 14, 64)	0
Conv2D	(None, 12, 12, 64)	36928
MaxPooling2 D	(None, 6, 6, 64)	0
Conv2D	(None, 4, 4, 64)	36928
MaxPooling2D	(None, 2, 2, 64)	0
Flatten	(None, 256)	0
Dense	(None, 64)	16448
Dense	(None, 10)	650
Nombre de paramètres		**184202**

TAB. 2.2 : Architecture du modèle proposé

2.6 Apprentissage par transfert (Transfer Learning)

Nous, les humains, sommes très doués pour appliquer le transfert de connaissances entre les tâches. Cela signifie que chaque fois que nous rencontrons un nouveau problème ou une nouvelle tâche, nous le reconnaissons et appliquons nos connaissances pertinentes issues de notre expérience d'apprentissage antérieure. Cela rend notre travail facile et rapide à terminer. Par exemple, si vous savez faire du vélo et que l'on vous demande de conduire une moto, ce que vous n'avez jamais fait auparavant. Dans ce cas, notre expérience de la bicyclette entrera en jeu et nous permettra de gérer des tâches telles que l'équilibre du vélo, la direction, etc. Cela facilitera les choses par rapport à un débutant qui n'a jamais fait de vélo. De tels réflexes sont très utiles dans la vie réelle car ils nous rendent plus performants et nous permettent d'acquérir plus d'expérience.

En suivant la même approche, un terme a été introduit : **Transfer Learning** dans le domaine de l'apprentissage profond. Cette approche implique l'utilisation des connaissances apprises dans une certaine tâche, et les appliquer pour résoudre une autre tache. Bien que des modèles soient conçus pour traiter une seule tâche, le développement d'algorithmes qui facilitent l'apprentissage par transfert est un sujet d'intérêt permanent dans la communauté de l'apprentissage automatique.

Pourquoi l'apprentissage par transfert ?

De nombreux réseaux de neurones profonds formés sur des images ont un phénomène particulier en commun : dans les premières couches du réseau, un modèle d'apprentissage profond tente d'apprendre un faible niveau de caractéristiques, comme la détection de contours, de couleurs, de variations d'intensité, etc. Ce type de caractéristiques ne semble pas être spécifique à un ensemble de données ou à une tâche particulière, quel que soit le type d'image que nous traitons, que ce soit pour détecter une feuille de plante ou des voitures. Dans les deux cas, nous devons détecter ces caractéristiques de bas niveau. Toutes ces caractéristiques apparaissent indépendamment de la fonction objectif Ainsi, l'apprentissage de ces caractéristiques dans une tâche de détection de feuille peut être utilisé dans d'autres tâches. C'est ce qu'est l'apprentissage par transfert. De nos jours, il est rare de former un réseau de neurones convolutifs entier à partir de zéro, mais il est courant d'utiliser un modèle pré-formé sur une variété d'images dans une tâche similaire, par exemple des modèles formés sur l'ensemble de données ImageNet [57](1,2 million d'images avec 1000 catégories), et d'utiliser les caractéristiques de celui-ci pour résoudre une nouvelle tâche.

La figure 2.15 montre le principe de l'apprentissage par transfert.

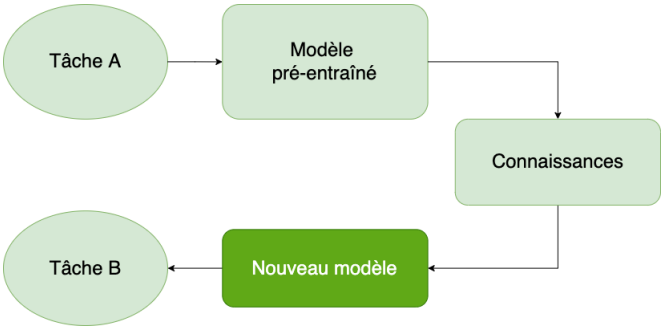


FIG. 2.15 : Diagramme montrant le processus simplifié d'apprentissage par transfert

Lorsqu'on traite de l'apprentissage par transfert, on rencontre un phénomène appelé " Freezing of layers ". Une couche, qu'il s'agisse d'une couche CNN, d'une couche cachée, d'un bloc de couches ou de tout sous-ensemble de toutes les couches, est dite fixe lorsqu'elle n'est plus disponible pour l'apprentissage. Par conséquent, les poids des couches gelées ne seront pas mis à jour pendant la formation. Alors que les couches qui ne sont pas gelées suivent la procédure de formation normale (voir figure 2.16).
Lorsque nous utilisons l'apprentissage par transfert pour résoudre un problème, nous choisissons un modèle pré-entraîné comme modèle de base. Maintenant, il y a deux approches possibles pour utiliser les connaissances du modèle pré-entraîné. La première consiste à fixer quelques couches du modèle pré-entraîné et à entraîner les autres couches sur notre nouvelle base de données pour la nouvelle tâche. La deuxième méthode consiste à créer un nouveau modèle, mais aussi à retirer certaines caractéristiques des couches du modèle pré-entraîné et à les utiliser dans un nouveau modèle. Dans les deux cas, nous retirons certaines des caractéristiques apprises et essayons d'entraîner le reste du modèle. Cela permet de s'assurer que la seule caractéristique qui peut être la même dans les deux tâches est retirée du modèle pré-entraîné, et que le reste du modèle est modifié pour s'adapter à la nouvelle base de donénes par l'entraînement [58].

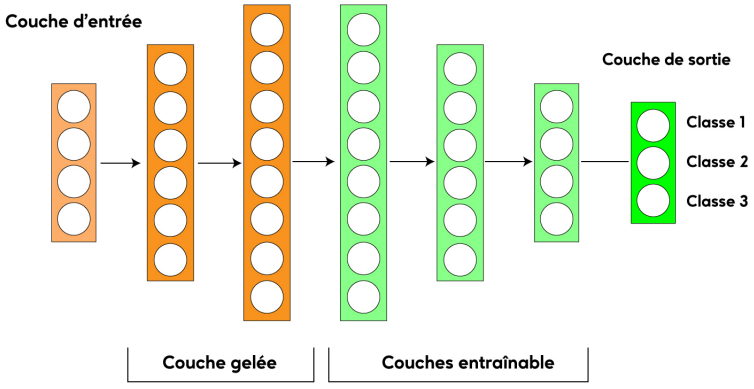


FIG. 2.16 : Couches fixes et les couches entraînables

L'idée est donc de fixer les poids de certaines couches pendant l'entraînement et d'af-

finer le reste pour faire face au problème. Cette stratégie permet de réutiliser les connaissances en termes d'architecture globale du réseau et d'utiliser son état comme point de départ pour la formation. Par conséquent, il peut obtenir de meilleures performances avec un temps d'entraînement plus court.
La figure 2.17 résume les principales méthodes d'apprentissage par transfert couramment utilisées en apprentissage profond :

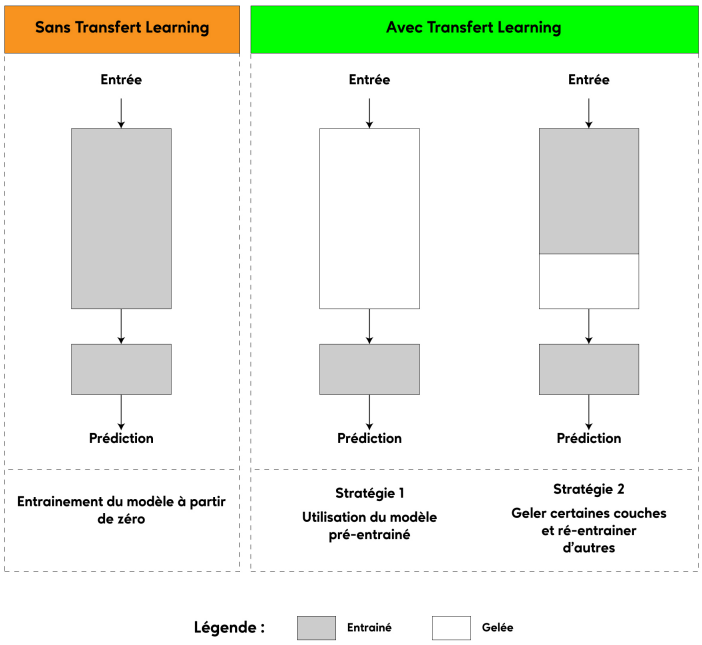


Fig. 2.17 : Approche d'apprentissage par transfert en Deep Learning

Par conséquent, l'une des exigences de base de l'apprentissage par transfert est l'existence d'un modèle performant sur la tâche source. Aujourd'hui, de nombreuses architectures d'apprentissage en profondeur à la pointe de la technologie sont désormais librement partagées par leurs équipes respectives.

2.7 Conclusion

Dans ce chapitre, nous avons commencé par évoquer l'histoire des réseaux neuronaux artificiels, puis nous avons détaillé les réseaux de neurones convolutifs et leurs principaux paramètres. Nous avons également présenté les architectures pré-entraînées qui sont le modèles CNN adoptés pour la classification des maladies des plantes.

Dans le chapitre suivant nous allons aborder la méthodologie suivie pour réaliser le système de diagnostic des maladies des plantes, ainsi que les résultats expérimentaux atteints et les différents analyses obtenus lors de l'utilisation des différentes types d'architectures CNN adaptées.

Chapitre 3

Méthodologie et résultats expérimentaux

3.1 Introduction

Ce chapitre est dédié à l'explication de la méthodologie de travail ainsi que l'analyse des performances des différents systèmes mis en œuvre pour l'identification de la classification des maladies agricoles. Nos tests ont été menés sur un sous-ensemble de la base PlantVillage [23] qui constitue un benchmark en agriculture intelligente. Plus précisément, la première partie des expériences examine et compare les architectures pré-entraînées en se basant sur l'apprentissage par transfert et en deuxième partie, nous évaluerons l'architecture CNN proposé pour le diagnostic des maladies des plantes.

L'implémentation des systèmes est effectuée sous environnement Anaconda du développement en Python. Comme l'apprentissage des CNN est une tâche très gourmande en capacités de calcul, l'exécution des programmes a été réalisée sur des machines virtuelles dans le cloud de Paperspace Gradient. Ce dernier est dédié aux applications deep learning qui requièrent l'utilisation de plusieurs bibliothèques spécifiques comme la Keras, PyTorch, et TensorFlow. La machine virtuelle est un *GPU* doté d'une carte graphique *Nvidia RTX5000* et 16GB de mémoire.

3.2 Ensemble de données

Tous les modèles CNN ont été entraînés sur un sous ensemble de 3 aliments (Potato, Tomato et Grape) avec 24 377 images. Ce dernier est issu d'une base de données accessible au public appelé Plant Village [23], qui contient un total de 54 306 images contenant 38 feuilles saines/malades différentes appartenant à 14 espèces végétales.
L'intégralité de cette base de données est disponible sur Kaggle [6], certaines des maladies végétales sont illustrées par la figure 3.1.

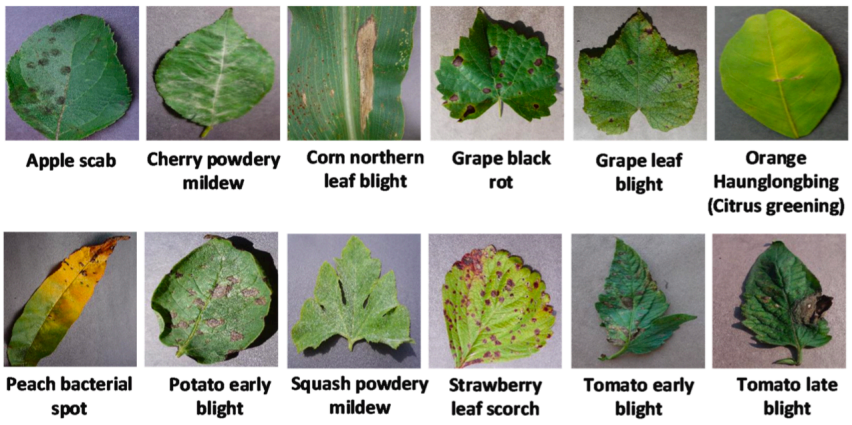


Fig. 3.1 : Quelques exemples des maladies des plantes de la base de données PlantVillage

Toutes les images de la base de données Plant Village ont été prises dans des stations de recherche expérimentale associées aux Land Grant Universities aux Etats-Unis (Penn State, Florida State, Cornell, et autres). La collecte des images continue, et à l'avenir, la liste de cette base de données va s'allonger. Les stations de recherche expérimentale

(publiques et privées) offrent la possibilité de prendre de nombreuses images en un temps réduit. La majorité des images ont été prises par deux techniciens travaillant en équipe. À partir d'essais en plein champ de cultures infectées par une maladie, les techniciens collectaient les feuilles en les enlevant de la plante. Les feuilles étaient ensuite placées sur une feuille de papier qui constituait un fond gris ou noir. Toutes les images ont été prises à l'extérieur, en pleine lumière. La lumière pouvait être un soleil fort ou des nuages et nous avons intentionnellement recherché un éventail de conditions car l'utilisateur final (cultivateur avec un smartphone) prendra finalement des images dans un éventail de conditions. Pour chaque feuille, 4 à 7 images sont collectées avec un appareil photo numérique standard (Sony DSC Rx100/13 20,2 mégapixels) en utilisant le mode automatique. La feuille a été tournée de 360 degrés pendant la prise de vue [23].

Distribution des échantillons

Dans ce travail, nous avons utilisé un sous-ensemble composé de 24 377 images de 3 types de plantes différentes à savoir : le raisin (Grape), la pomme de terre (Potato) et la tomate (Tomato). L'analyse multi-classe des pathologies a été menée sur les infections de ces 3 plantes tel que Grape compte 4 classes, Potato compte 3 classes, et Tomato compte 10 classes. Le nombre d'échantillons par classe est illustré dans le tableau 3.1.

Type d'aliments	Maladie associée	Nombre d'images
Grape	Black rot	1180
	Esca (Black Measles)	1383
	Leaf blight (Isariopsis Leaf Spot)	1076
	Healthy	423
Potato	Early blight	1000
	Late blight	1000
	Healthy	152
Tomato	Bacterial spot	2127
	Early blight	1000
	Late blight	1909
	Leaf Mold	952
	Septoria leaf spot	1771
	Spider mites Two-spotted spider mite	1676
	Target Spot	1404
	Yellow Leaf Curl Virus	5357
	Mosaic virus	373
	Healthy	1590
Nombre de classes	17	24373

Tab. 3.1 : Répartition des échantillons de l'ensemble de donnée Plant Village

On remarque que l'ensemble de données est très déséquilibré c'est à dire qu'il existe des classe représentées avec plus d'instances que d'autres classes. Dans ce cas, les modèles CNN ont tendance à créer un modèle d'apprentissage biaisé qui a une moins bonne précision prédictive sur les classes minoritaires par rapport aux classes majoritaires. Dans de rares cas, comme la détection des fraudes ou la prédiction des maladies, comme dans notre cas,

il est vital d'identifier correctement les classes minoritaires. Le modèle ne doit donc pas être biaisé pour ne détecter que la classe majoritaire, mais doit accorder le même poids ou la même importance à la classe minoritaire. Par conséquent, le nombre de données par classe doit augmenter. Aujourd'hui, une technique très populaire appelé "augmentation de données" ou data augmentation est utilisée pour augmenter la quantité de données dans l'ensemble d'apprentissage en ajoutant des copies légèrement modifiées de données déjà existantes ou des données synthétiques nouvellement créées à partir de données existantes. Cette technique est expliquée dans ce qui suit.

Augmentation de données ou data augmentation

L'augmentation de données permet d'augmenter la diversité de notre ensemble d'entraînement en appliquant des transformations aléatoires (mais réalistes), telles que (rotation d'image, inversion, zoom) comme illustré par la figure 3.2.

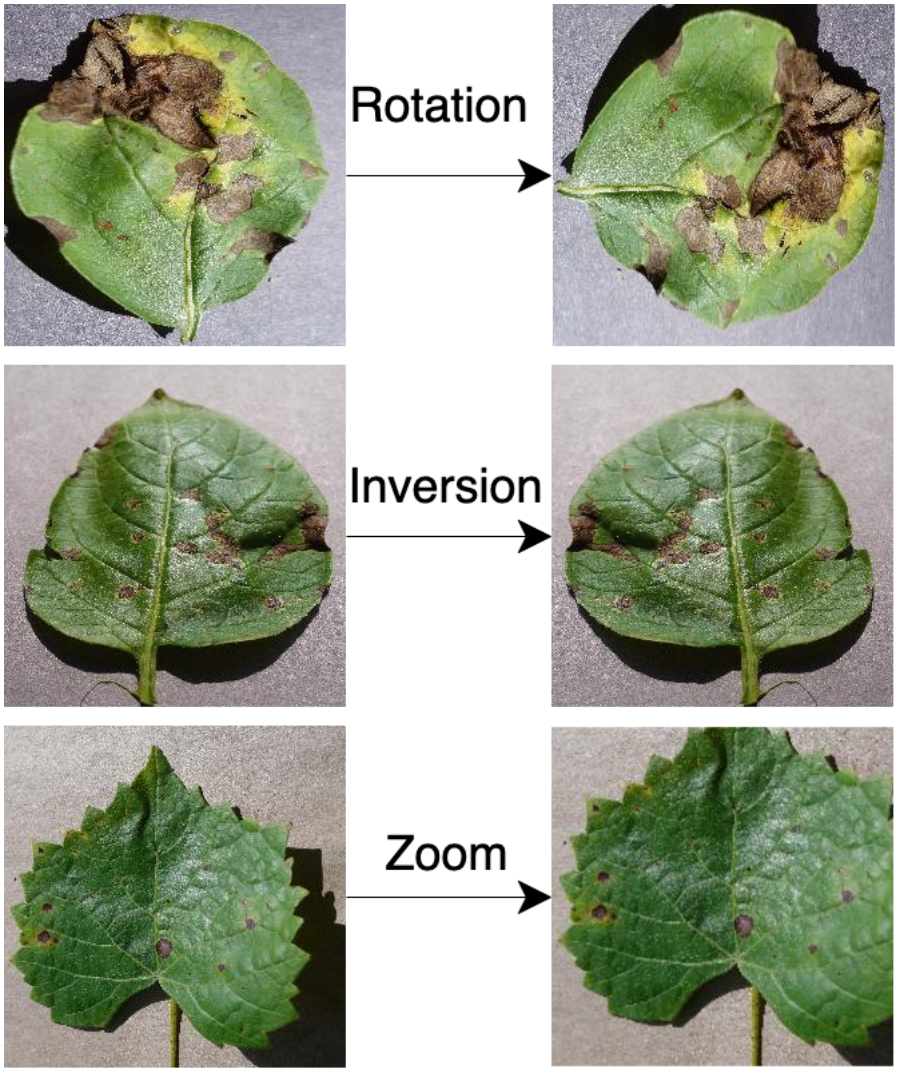


Fig. 3.2 : Examples de transformation faites sur les images

Le tableau 3.2 montre la distribution des échantillons dans les 17 classes après l'augmentation des données. Notons que 80% des échantillons sont utilisés pour la phase d'apprentissage, 10% pour la validation et 10% pour la phase de test.

Type d'aliment	Maladie associée	Entraînement	Validation	Test	Totale
Grape	Black rot	944	118	118	1180
	Esca (Black Measles)	1106	138	139	1383
	Leaf blight (Isariopsis Leaf Spot)	860	107	109	1076
	Healthy	800	100	100	1000
Potato	Early blight	800	100	100	1000
	Late blight	800	100	100	1000
	Healthy	800	100	100	1000
Tomato	Bacterial spot	1701	212	214	2127
	Early blight	1920	240	240	2400
	Late blight	1851	231	232	2314
	Leaf Mold	1881	235	236	2352
	Septoria leaf spot	1744	218	219	2181
	Spider mites Two-spotted spider mite	1740	217	219	2176
	Target Spot	1827	228	229	2284
	Yellow Leaf Curl Virus	1960	245	246	2451
	Mosaic virus	1790	223	225	2238
	Healthy	1925	240	242	2407
Total	**17**	**24449**	**3052**	**3068**	**30569**

TAB. 3.2 : Répartition des échantillons de notre ensemble de données après l'augmentation de données

3.3 Logiciels, libraires et matériels

Dans cette section, nous donnons un bref aperçu du langage de programmation et des outils logiciels utilisés dans notre travail, à savoir Python, Keras, TensorFlow ainsi que du matériel utilisé.

3.3.1 Logiciels, libraires

Python

Python est un langage de programmation interprété, orienté objet, de haut niveau et doté d'une sémantique dynamique. Ses structures de données intégrées de haut niveau, combinées au typage dynamique et à la liaison dynamique, le rendent très attrayant et polyvalent qui peut être utilisé dans de nombreux domaines, tels que le développement Web, l'analyse de données, l'apprentissage profond et de l'intelligence artificielle. Python supporte les modules et les packages, ce qui encourage la modularité des programmes et la réutilisation du code. L'interpréteur Python et sa vaste bibliothèque standard sont disponibles gratuitement pour toutes les principales plates-formes et peuvent être distribués librement [59]. On a travaillé avec la version : 3.9.12.

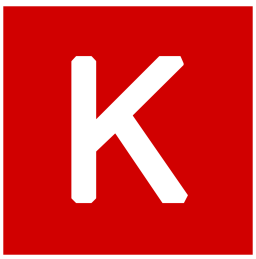
Conda

Conda est un système de gestion de paquets et d'environnement open source qui fonctionne sous Windows, macOS, Linux et z/OS. Conda installe, exécute et met à jour rapidement les paquets et leurs dépendances. Conda crée, enregistre, charge et bascule facilement entre les environnements de notre ordinateur. Il a été créé pour les programmes Python, mais il peut empaqueter et distribuer des logiciels pour n'importe quel langage [80]. On a travaillé avec la version : 4.13.

Tensorflow

TensorFlow est une plateforme open-source développée par Google (end to end) pour la création d'applications d'apprentissage automatique. Il s'agit d'une bibliothèque de mathématiques dédiée au calcul numérique de haute performance, qui utilise le flux de données et la programmation différentiable pour effectuer diverses tâches axées sur la formation et l'inférence de réseaux de neurones profonds. Elle permet aux développeurs de créer des applications d'apprentissage automatique en utilisant divers outils, bibliothèques et ressources communautaires. On a travaillé avec la version : 2.9.1

Keras



Keras est une bibliothèque python open source de haut niveau mise en œuvre pour prendre en charge les réseaux neuronaux profonds. Elle fournit aux utilisateurs une interface pratique pour manipuler ces derniers en tirant parti d'autres bibliothèques telles que Theano, CNTK et TensorFlow. Pour notre application, nous avons associé la version 2.9 Keras à TensorFlow.

Scikit-Learn

Scikit-learn est une bibliothèque Python gratuite pour l'apprentissage automatique. Il est développé par de nombreux contributeurs, notamment universitaires, par des instituts d'enseignement supérieur et de recherche français comme Inria [60].

Nous l'avons principalement utilisée pour la création et la visualisations des modèles. La principale caractéristique de cette API est de permettre des tracés rapides et des ajustements visuels sans refaire les calculs.

3.3.2 Matériel

Pour le matériel utilisé nous avons mené tous les entraînements sur le cloud pour ganger du temps vu que les CNN demandent beaucoup de ressources matérielles alors on a opté pour la plateforme Paperspace Gradient. Entraîné un CNN nécessite énormément de ressources matérielles, nous avons donc opté pour l'utilisation d'un Cloud de la plateforme Paperspace Gradient.

Paperspace Gradient

Paperspace Gradient est une plateforme contemporaine qui vise à apporter la simplicité et la flexibilité pour construire des modèles d'apprentissage automatique dans le cloud, qui expose des VM (Machine Virtuelle) brutes alimentées par des GPU NVIDIA et des TPU Google aux clients qui préfèrent un contrôle supplémentaire sur l'infrastructure et pour accélérer les entraînements de leurs modèles [61].
Cette plateforme est payante, en ce qui concerne les offres d'abonnement disponibles, nous avons choisi l'offre PRO à 8$ avec un accès à 2 GPU performants de Nvidia dont les caractéristiques sont citées dans le tableau ci-dessous :

Nvidia Quadro RTX5000	Cœurs de traitement parallèle CUDA	3072
	Cœurs NVIDIA Tensor	384
	Cœurs NVIDIA RT	48
	Mémoire GPU	**16 Go GDDR6**
	RTX-OPS	62T
	Ray casting	8 Giga Rays/sec
	Performances FP32	11,2 TFlops
	Bus graphique	PCI Express 3.0 x16
	NVLink	Oui
	Connecteurs d'affichage	DP 1.4 (x4), VirtualLink (x1)
	Configuration	11 cm (H) X 26,5 cm (L), Dual Slot
	VR Ready	Oui
Nvidia Quadro RTX4000	Cœurs de traitement parallèle CUDA	2304
	Cœurs NVIDIA Tensor	288
	Cœurs NVIDIA RT	36
	Mémoire GPU	**8 Go GDDR6**
	RTX-OPS	43T
	Ray casting	6 Giga Rays/sec
	Performances FP32	7,1 TFlops
	Consommation maximale	160 W
	Bus graphique	PCI Express 3.0 x16
	Connecteurs d'affichage	DP 1.4 (x3), VirtualLink (x1)
	Configuration	11 cm (H) X 24 cm (L), Single Slot
	VR Ready	Oui

TAB. 3.3 : Caractéristiques des GPU utilisées dans les entraînements

3.4 Métriques d'évaluation

Les métriques d'évaluation sont utilisées pour mesurer la qualité du modèle. L'un des sujets les plus importants de l'apprentissage profond est la manière d'évaluer notre modèle. Lorsque l'on construit un modèle, il est très important de mesurer la précision avec laquelle il prédit le résultat attendu.
Si l'on ne procède pas à une évaluation correcte du modèle les métriques d'évaluation adéquates, cela peut aboutir à de mauvaises prédictions surtout si l'ensemble de données est déséquilibré.
De ce fait, nous proposons d'utiliser différentes métriques d'évaluation présentées dans ce qui suit.

3.4.1 Précision de la classification (Classification Accuracy)

La métrique la plus simple pour l'évaluation des modèles est la précision. Il s'agit du rapport entre le nombre de prédictions correctes et le nombre total de prédictions effectuées pour un ensemble de données .

$$Précision\ (\%) = \frac{Nombre\ de\ prédictions\ correctes}{Nombre\ total\ de\ prédictions\ effectuées} \times 100 \tag{3.1}$$

Cette métrique est utilisée lorsque l'ensemble de données est équilibré.

3.4.2 Matrice de confusion(Confusion Matrix)

Une matrice de confusion ou matrice d'erreurs est un tableau qui montre le nombre de prédictions correctes et incorrectes faites par le modèle par rapport aux classifications réelles un ensemble de données, nous renseignant ainsi sur les confusions (erreurs)faites par le modèle.
Cette matrice décrit la performance d'un modèle de classification sur des données test pour lesquelles les vraies valeurs sont connues. Il s'agit d'une matrice $n \times n$, où n est le nombre de classes. La figure 3.3 montre une matrice de confusion pour un problème à 2 classes. Comme on peut le voir, les résultats d'une matrice de confusion sont classés en quatre grandes catégories : les vrais positifs (VP), les vrais négatifs (VN), les faux positifs (FP) et les faux négatifs (FN).

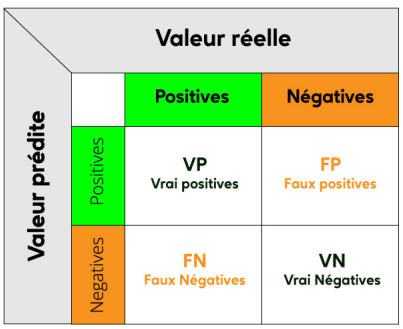


Fig. 3.3 : Modèle de base de la matrice de confusion

- **Les vrais positifs (VP) :**Nombre d'échantillons qui sont réellement positifs et qui sont prédits positifs.
- **Vrais négatifs (VN) :** Nombre d'échantillons qui sont réellement négatifs et qui sont prédits négatifs.
- **Faux positifs (FP) :** Nombre d'échantillons qui sont en réalité négatifs mais prédits positifs. Ces erreurs sont également appelées erreurs de type 1.
- **Faux négatifs (FN) :** Nombre d'échantillons qui sont en fait positifs mais prédits négatifs. Ces erreurs sont également appelées erreurs de type 2.

Nous pouvons obtenir 4 mesures de classification à partir de la matrice de confusion :

Précision

La précision répond à la question suivante : Quelle proportion de prédictions positives est réellement correcte ? Elle définit alors le nombre de vrais positifs par rapport aux nombres de vrais positifs et faux positifs, tel que :

$$Précision\ (\%)\ = \frac{VP}{VP\ +\ FP} \times 100 \tag{3.2}$$

Recall ou Sensibilité ou Rappel

Le recall répond à la question suivante : quelle proportion de vrais positifs a été identifiée correctement ? Il représente alors le rapport entre le nombre total d'exemples positifs correctement classés et le nombre total d'exemples positifs. L'équation peut être énoncée comme suit :

$$Recall\ (\%) = \frac{VP}{VP+FN} \times 100 \tag{3.3}$$

F1-score

Le score F1 donne une estimation globale de la précision et du rappel d'un échantillon. Il s'agit de la moyenne harmonique de la précision et du rappel d'un échantillon, le score F1 peut être défini comme suit :

$$F1 - score(\%) = \frac{2\times\ Précision\ \times\ Recall}{Précision\ +\ Recall} \times 100 \tag{3.4}$$

3.5 Protocole expérimental

Le développement d'un système de diagnostic des maladies des plantes nécessite un protocole expérimental bien précis afin d'aboutir à des résultats satisfaisants. Dans cette section, nous décrivons le protocole expérimental adopté en terme de méthodologie, de répartition des données et de paramétrage des architectures CNN.

3.5.1 Méthodologie

Notre objectif principal est de développer éventuellement un modèle pour identifier 17 classes de feuilles de plantes en fonction du type d'aliment et des maladies associées à cet aliment. Pour cela nous utiliserons deux méthodes :

Pour rappel, dans ce travail, nous nous intéressons à 3 plantes c'est à dire à 3 aliments et nous comparons deux méthodes pour aboutir au diagnostic d'une feuille de plante. Tout d'abord, nous faisons une classification en fonction du type d'aliment, puis une classification selon les maladies associées à cet aliment. Ainsi, pour cette première méthode, nous avons 4 modèles de classification.
La deuxième méthode, crée un modèle pour les 17 classes à la fois, qui est plus optimal pour l'application Web que nous présenterons dans le chapitre suivant. La figure 3.4 schématise les deux méthodes de classification des maladies des plantes.

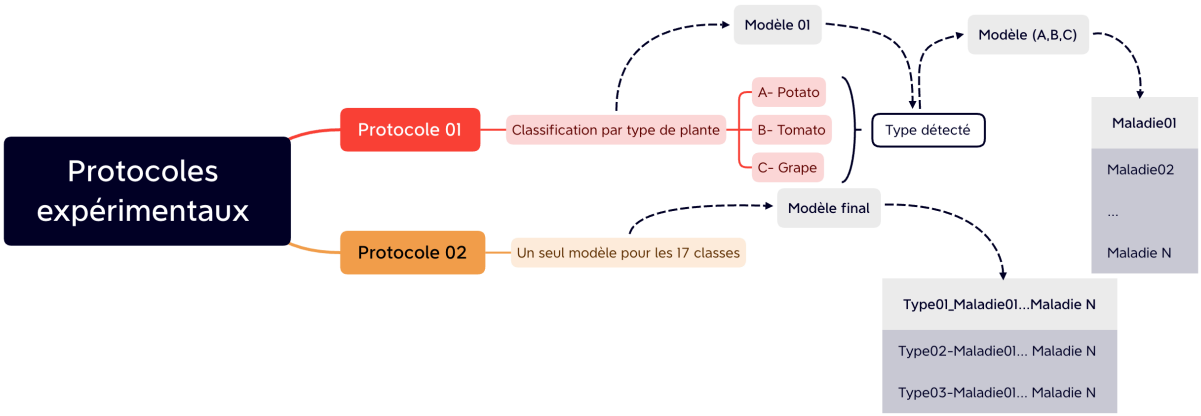


FIG. 3.4 : Méthodes adoptées pour le diagnostic des maladies des plantes

3.5.2 Répartition des données

Les images récoltées de la base de données Plant Village sont réparties selon selon les deux protocoles expliqués précédemment.

Protocole 01

Pour la classification par type de plantes c'est à dire potato, grape ou tomato, nous avons pris des échantillons de chaque classe d'aliment afin qu'il y ait le plus de diversité possible. Sachant que potato contient 3 classes, avec 2155 images. Tandis que grape contient 4 classes (630 échantillons ont été pris de chaque classe) cumulant un total de 2524 images. Enfin tomato contient 10 classes avec 290 échantillons par classe donnant un total de 2900 images. Notons que 80% des échantillons sont utilisés pour la phase d'entraînement du modèle, 10% pour la validation et 10% pour le test. Le tableau 3.4 résume cette répartition.

	Potato	Tomato	Grape
Entraînement 80%	1724	2320	2019
Validation 10%	215	290	252
Test 10%	216	290	256
Total	2155	2900	2527

Tab. 3.4 : Répartition des données pour la classification du type de plante

Par la suite, pour la classification des maladies de chaque type de plante, les données ont été répartis dans les 3 ensembles ; entraînement, validation et test selon les tableaux 3.6, 3.7, 3.5.

	Entraînement	Validation	Test
Early blight	800	100	100
Late blight	800	100	100
Healthy	800	100	100
Totale	2400	300	300

Tab. 3.5 : Répartition des données
Type : Potato

	Entraînement	Validation	Test
Bacterial spot	1701	212	214
Early blight	1920	240	240
Late blight	1851	231	232
Leaf Mold	1881	235	236
Septoria leaf spot	1744	218	219
Spider mites Two-spotted spider mite	1740	217	219
Target Spot	1827	228	229
Yellow Leaf Curl Virus	1960	245	246
Mosaic virus	1790	223	225
Healthy	1925	240	242
Totale	18339	2289	2302

Tab. 3.6 : Répartition des données
Type : Tomato

	Entraînement	Validation	Test
Black rot	944	118	118
Esca (Black Measles)	1106	138	139
Leaf blight (Isariopsis Leaf Spot)	860	107	109
Healthy	800	100	100
Totale	3710	463	466

TAB. 3.7 : Répartition des données
Type : Grape

Protocole 02

Cette fois-ci une classification des 17 classes à la fois est réalisée en utilisant la répartition des données indiquée dans le tableau suivant :

	Entraînement	Validation	Test
Grape Black rot	944	118	118
Grape Esca(Black Measles)	1106	138	139
Grape Leaf blight (Isariopsis Leaf Spot)	860	107	109
Grape Healthy	800	100	100
Potato Early blight	800	100	100
Potato Late blight	800	100	100
Potato Healthy	800	100	100
Tomato Bacterial spot	1701	212	214
Tomato Early blight	1920	240	240
Tomato Late blight	1851	231	232
Tomato Leaf Mold	1881	235	236
Tomato Septoria leaf spot	1744	218	219
Tomato Spider mites Two-spotted spider mite	1740	217	219
Tomato Target Spot	1827	228	229
Tomato Tomato Yellow Leaf Curl Virus	1960	245	246
Tomato Tomato mosaic virus	1790	223	225
Tomato Healthy	1925	240	242
Totale	24449	3052	3068

TAB. 3.8 : Répartition des données des 17 classes

3.5.3 Entraînement

- La construction d'un modèle efficace nécessite d'analyser à la fois la conception du réseau mais aussi le format des données d'entrée. Ainsi les images des feuilles de plantes ont été pré-traitées afin que le modèle CNN puisse extraire les caractéristiques appropriées de l'image. De ce fait, deux prétraitements en été réalisés sur notre ensemble de données a savoir ; normalisation des valeurs RVB entre 0 et 1, et redimensionnement de la taille des images en fonction du modèle utilisé.

- Il était difficile de charger toutes les images avec une taille de 256 x 256, donc le générateur de données est utilisé pour réduire l'utilisation de la mémoire.

- Lors de la phase d'entraînement, nous avons utilisé les poids d'ImageNet pour chacune des architectures pre-entraînées. La taille des images de feuilles en entrée dépend du modèle que nous utilisons. L'ensemble de données dont nous disposons a été divisé 3 sous-ensembles ; entraînement, validation et test, avec un pourcentage de 80%, 10% et 10%, respectivement comme il est montré dans les tableaux précédents.Le sous-ensemble d'entraînement fait référence aux données utilisées pour former le modèle d'apprentissage profond. Le sous-ensemble de données de validation est quant à lui utilisé pour mesurer la perfection du modèle DL pendant l'entraînement. Enfin, les données de test sont considérées comme l'ensemble final de données utilisé pour mesurer les performances du modèle sur des données entièrement inédites.

- Quant à la mise a jour des poids du réseaux entièrement connecté dans le CNN, les optimiseurs Adam et SGD (pour Stochastic Gradient Descent) sont utilisés avec différentes valeurs du taux d'apprentissage pour chaque modèle CNN comme indiqué dans le tableau 3.9. Une taille de batch de 32 est utilisée pour l'entraînement de tous les modèles présentés dans la section 2.5.

Le tableau 3.9 récapitule tous les paramètres utilisés lors de l'entraînement en fonction du modèle.

	Architecture			
Paramètres d'entraînement	**AlexNet**	**VGG16**	**ResNet152v2**	**Notre modèle**
Taille de l'image d'entrée en pixels	64×64	224×224	256×256	256×256
Nombre de couches de convolution	3	13	150	6
Nombre de couches de max-pooling.	3	5	2	6
Valeur du Dropout	0.2, 0.2,0.4 (Conv)	0.25 (Dense)	Non	Non
Poids du réseau attribué	Imagenet	Imagenet	Imagenet	Non
Fonction d'activation	Relu, sortie (Softmax)	Relu, sortie (Softmax)	Relu, sortie (Softmax)	Relu, sortie (Softmax)
Taux d'apprentissage	0,0005	0,0001	0,01	0,001
Epochs	75	25	5	50
Batch Size	32	32	32	32
Optimiseur	Adam	SGD	SGD	Adam
Type de Classifieur	Couche entièrement connectée (ANN)	Couche entièrement connectée (ANN)	Couche entièrement connectée (ANN)	Couche entièrement connectée (ANN)

Tab. 3.9 : Paramètres d'entraînement adoptés

3.6 Performances atteintes

Dans un premier temps, nous allons évaluer les différents modèle CNN adoptés pour le diagnostic des maladies des plantes selon les 2 protocoles présentés précédemment.

3.6.1 Protocole 01

Rappelons que dans ce premier protocole, nous procédons d'abord à une classification par type de plante avant de passer à la classification de la maladie que présente la feuille selon le type de plante trouvé. Pour cela, plusieurs critère d'évaluation sont pris en compte notamment ; la taille du modèle, le temps nécessaire à son entraînement ainsi que temps d'inférence d'une seule donnée puisque nous envisageons une implémentation Web du système final. L'évaluation quantitative de la performance des différents modèles est principalement basée sur la précision (accuracy) dans les différents ensembles entraînement, validation et test.

Les résultats de la classification par type de plantes sont donnés dans le tableau 3.10. Nous remarquons,

Detection du type de plante						
	Taille du modèle	**Précision d'entraînement (%)**	**Précision de validation (%)**	**Précision de test (%)**	**Temps d'entraînement**	**Temps d'inference d'une seule Image**
AlexNet	4.9 Mo	99,90	99,73	99,37	5min (50 epochs)	**0.7s**
VGG16	180.6 Mo	100,00	99,86	**100,00**	4min30s (15 epochs)	0.8s
ResNet152v2	203 Mo	100,00	99,73	**100,00**	**2min30s (10 epochs)**	1.75s
Notre modèle	**2.3 Mo**	100,00	99,84	99,75	4min20s (50 epochs)	**0.7s**

TAB. 3.10 : Résultats de la classification par type de plante

Les résultats de la classification du type de plante montre une précision parfaite de 100% pour les modèles VGG16 et ResNet152 suivis du modèle proposé qui présente une précision de 99.75%. Aussi, les temps d'entraînement restent raisonnables avec le temps le plus court de 2min30s avec 10 epochs offert par ResNet152. En 2ème position, vient le modèle proposé avec un temps d'entraînement de 4min20 avec 50 epochs. Quant au temps d'inférence, notre modèle ainsi que AlexNet offre le temps le plus court qui est de 0.7s pour une seule image. ResNet152 présente un temps d'inférence relativement élevé, 1.75s soit 2min55s. Par ailleurs et sans surprise, ResNet152 et VGG16 occupent le plus d'espace mémoire avec une taille de 203Mo et 180.6Mo, respectivement. Le modèle proposé ainsi que AlexNet sont les moins gourmands en espace mémoire car ils sont les moins profonds. Globalement, le modèle proposé présente les meilleurs performances, même s'il présente une erreur de 0.25% sur les données test qui correspond à 2 images incorrectement classées sur un total de 762 images.

Pour la deuxième partie de cette expérience qui concerne la classification des maladies associées à chaque type d'aliment, les résultats obtenus sont illustrés dans le tableau 3.11 :

Potato						
	Taille du modèle	**Précision d'entraînement (%)**	**Précision de validation (%)**	**Précision de test (%)**	**Temps d'entraînement**	**Temps d'inference d'une seule Image**
AlexNet	4.9 Mo	99,54	98,67	96,67	5min50s (70 epochs)	0.2s
VGG16	169Mo	99,67	100,00	99,67	8min42s (20 epochs)	0.5s
ResNet152v2	243. 1Mo	99,87	100,00	**100,00**	**3min2s (5 epochs)**	1.7s
Notre modèle	**2.3 Mo**	99,33	100,00	98,33	21min20s (40 epochs)	**0.1s**
Tomato						
	Taille du modèle	**Précision d'entraînement (%)**	**Précision de validation (%)**	**Précision de test (%)**	**Temps d'entraînement**	**Temps d'inference d'une seule Image**
AlexNet	4.9 Mo	99,19	96,68	97,65	23min45s (75 epochs)	**0.2s**
VGG16	169 Mo	99,90	98,43	98,57	46min33s (22 epochs)	0.9s
ResNet152v2	243 Mo	99,79	99,00	**99,26**	**19min18s (5 epochs)**	1.7s
Notre modèle	**2.1 Mo**	99,19	95,09	96,78	30min 50s (50 epochs)	0.7s
Grape						
	Taille du modèle	**Précision d'entraînement (%)**	**Précision de validation (%)**	**Précision de test (%)**	**Temps d'entraînement**	**Temps d'inference d'une seule Image**
AlexNet	4.9 Mo	99,73	99,35	99,14	5min1s (70 epochs)	**0.2s**
VGG16	169 Mo	100,00	99,78	**99,57**	8min45s (20 epochs)	**0.2s**
ResNet152v2	243 Mo	99,92	99,57	**99,57**	4min13s (5 epochs)	1.8s
Notre modèle	**2.3 Mo**	100,00	99,14	**99,57**	**3min 37s (30 epochs)**	**0.2s**

Tab. 3.11 : Résultats de la classification multiclasse

En analysant le tableau 3.11, nous remarquons que le modèle ResNet152 offre généralement le temps d'entraînement le plus court et la précision la plus élevée pour les 3 aliments à savoir potato qui possède 3 classes, tomato qui possède 10 classes et grape qui possède 4 classes. Par ailleurs notre modèle propose une performance très satisfaisante avec des taux d'erreur allant de 0.4% à 3.2% pour tomato. Cependant, le modèle basé sur ResNet152 présente un temps d'inférence le plus élevé autour de 1.7s comparativement a notre modèle qui répond plus rapidement. Quant à la taille des modèles, encore une fois ResNet152 occupe le plus d'espace mémoire suivi de VGG16 et d'AlexNet. Le modèle proposé quant à lui possède une taille de 2.3Mo ce qui est très appréciable pour une implémentation Web.

3.6.2 Protocole 02 : 17 classes en même temps

Pour ce deuxième protocole, nous avons abordé la classification des maladies en faisant combiner les 3 types de plantes à la fois, pour en faire qu'un seul modèle capable de distinguer entre 17 classes (10 classes Tomato, 4 classes Grape et 3 classes Potato). Pour les critères d'évaluation des performances des modèles, nous avons utilisé, la précision, le recall, le F1-score ainsi que l'analyse de la matrice de confusion pour comprendre de quelle façon le modèle est confus lorsqu'il effectue des prédictions. Non seulement cela nous permettra de savoir quelles erreurs sont commises, mais surtout, les types de confusion précisément par rapport à l'aliment et aux différentes maladies. Les résultats obtenus sont

résumés dans le tableau 3.12.

	Taille du modèle	Précision d'entraînement (%)	Précision de validation (%)	Précision de test (%)	Temps d'entraînement	Temps d'inference d'une seule Image
AlexNet	4.9 Mo	98,86	96,53	95,99	31min (70 epochs)	0.7s
VGG16	184.6 Mo	99,95	98,69	98,83	72min54s (25 epochs)	0.8s
ResNet152v2	243 Mo	99,83	99,18	**99,28**	**25min30s (5 epochs)**	1.7s
Notre modèle	**2.3 Mo**	98,91	96,63	95,58	39min 20s (50 epochs)	**0.7s**

TAB. 3.12 : Résultats de classification des 17 classes en même temps

Dans cette expérience ou nous avons à la fois le type d'aliment et sa maladie, ResNet152 atteint une précision de 99.28% suivi de VGG16 avec 98.83%. Notre modèle vient en dernière position avec une précision globale de 95.58% offrant une erreur de 4.42% ce qui correspond à 136 images mal classées sur un total de 3068 images. De plus, en terme de temps d'entraînement ResNet152 apprend au bout de 25min30s (5epochs) alors que notre modèle prend un peu plus de temps avec 39min24s (50 epochs). Cependant, le temps d'inférence de ResNet152 est plus de deux fois plus long que le temps offert par notre modèle qui est de 0.7s. Afin de mieux apprécier les performances de notre modèle, nous présentons la matrice de confusion (figure 3.5) ainsi que les métriques précision, recall et F-1 score par classe (tableau 3.13)

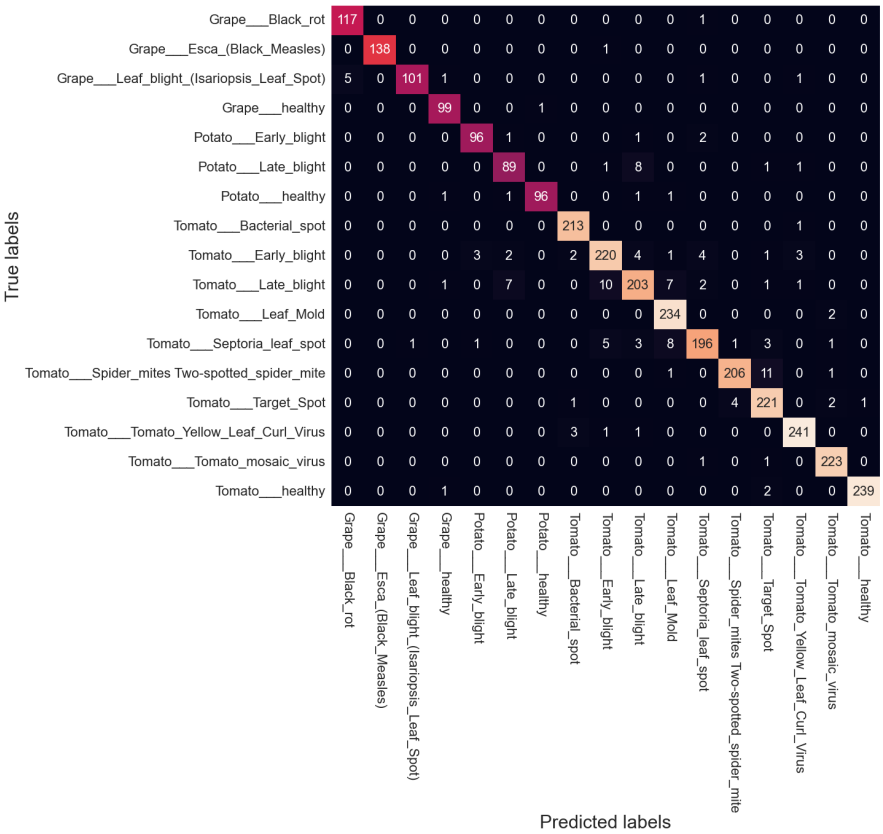


FIG. 3.5 : Matrice de confusion du modèle proposé

L'analyse de la matrice de confusion, montre une confusion entre potato et tomato pour la même maladie "late blight". En effet, 8 images sur 100 de potato late blight ont été classées tomato late blight, et 7 images sur 232 tomato late blight ont été classées potato late blight. Aussi, nous remarquons une confusion importante dans les maladies de la plante tomato, notamment entre les maladies late blight, early blight, septoria leaf spot, leaf mold, spider mites two spotted spider mite et target spot, telles que :
- 10 images tomato late blight sont classées tomato early blight
- 8 images de tomato septoria leaf spot sont classées tomato leaf mold
- 11 images de tomato spider mites two spotted spider mite sont classées tomato target spot

Cette analyse peut etre confirmée par les valeurs de la précision et du recall données dans le tableau 3.13, plus la précision est élevée plus le modèle minimise le nombre de faux positifs, tandis que plus le recall est élevé, plus le modèle maximise le nombre de vrais positifs. Pour avoir une vue plus globale sur la performance du modèle nous considérons le F-1 score, plus il est élevé plus le modèle est performant.
Les meilleurs résultats atteints par le modèle CNN proposé sont récapitulés dans la figure 3.6

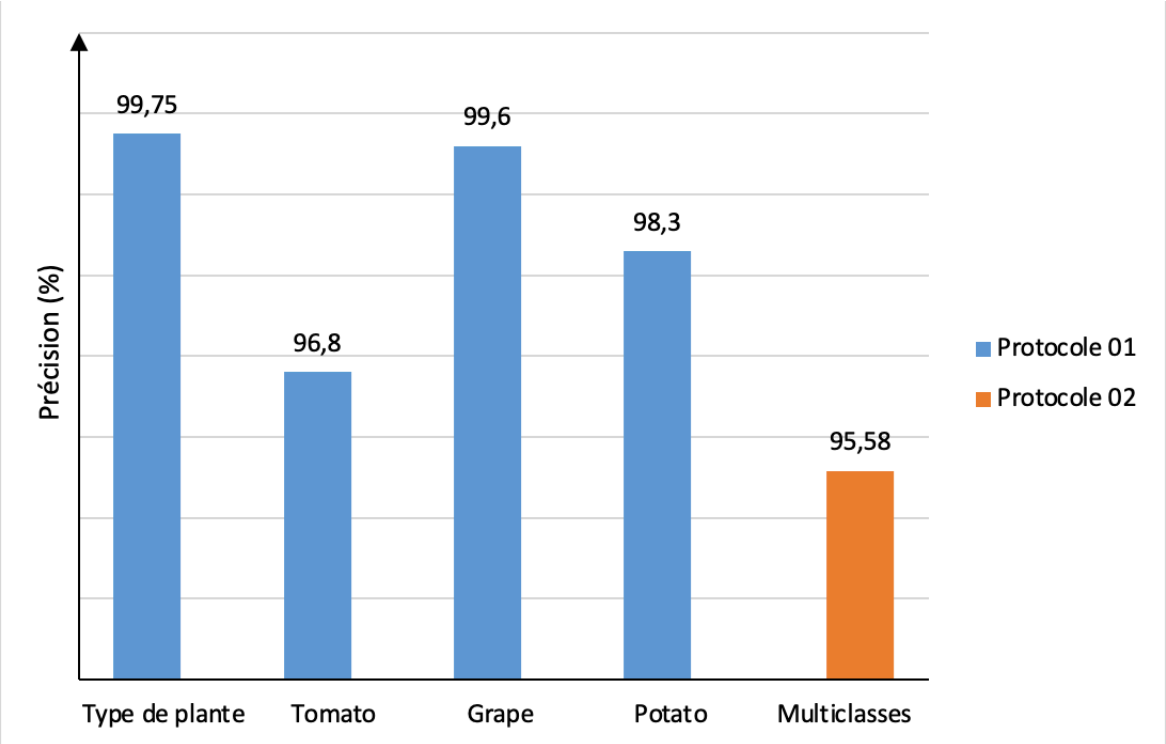


FIG. 3.6 : Comparaison entre les résultats obtenus au niveau des deux protocoles

	Précision (%)	Recall (%)	F1-score (%)	Nombre d'images
Black rot	95,90	99,15	97,50	118
Esca (Black Measles)	100,00	99,28	99,64	139
Leaf blight (Isariopsis Leaf Spot)	99,02	92,66	95,73	109
Healthy	96,12	99,00	97,54	100
Early blight	96,00	96,00	96,00	100
Late blight	89,00	89,00	89,00	100
Healthy	98,97	96,00	97,46	100
Bacterial spot	97,26	99,53	98,38	214
Early blight	92,44	91,67	92,05	240
Late blight	91,86	87,50	89,62	232
Leaf Mold	92,86	99,15	95,90	236
Septoria leaf spot	94,69	89,50	92,02	219
Spider mites Two-spotted spider mite	97,63	94,06	95,81	219
Target Spot	91,70	96,51	94,04	229
Yellow Leaf Curl Virus	97,18	97,97	97,57	246
Mosaic virus	97,38	99,11	98,24	225
Healthy	99,58	98,76	99,17	242
accuracy	95,57	95,57	95,57	
macro avg	95,74	95,58	95,63	3068
weighted avg	95,59	95,57	95,55	3068

Tab. 3.13 : Précision, recall et F1-score par classe du modèle proposé

3.7 Discussion

Les tests effectués pour le développement d'un système de diagnostic des plantes malades ont porté sur l'utilisation des réseaux de neurones convolutifs. A partir des tests menés sur un sous-ensemble de la base Plant Village selon 2 protocoles, nous notons :

- Les architectures pré-entrainées telles que ResNet152v2 et VGG16 offrent les meilleurs performances, cependant, leur utilisation requiert des capacités de calculs très élevées ce qui nécessite l'utilisation de beaucoup de ressources matérielles (GPU), ce qui nous a amené à utiliser les machines virtuelles offertes par la plateforme Paperspace Gradient. De plus, ces modèles occupent beaucoup d'espace mémoire à cause de leurs architectures complexes, ce qui n'est pas rentable pour une implémentation web de la solution proposée. Ces raisons nous ont poussé à les laisser de coté malgré leurs prouesses, et a chercher a développer une architecture plus simple et moins gourmande en temps et en ressources tout en offrant des performances satisfaisantes.

- Le modèle CNN proposé est caractérisé par une taille relativement petite par rapport aux architectures pré-entraînées, et ce tout en gardant des précisions élevées. D'autre part, les expériences faites sur les deux architectures pré-entraînées VGG16 et AlexNet ont démontré l'importance du Dropout et l'ajout des couches cachées dans l'amélioration du pouvoir de classification des CNN.

- Nous constatons que le modèle mis en oeuvre parvient à maintenir leur pouvoir d'isolation entre les classes associées à chaque type de plante, la précision sur les données test oscille entre 96% et 99%. Il est à noter que la tache de séparation entre les trois de plantes au niveau du protocole 01 est relativement simple. Nous remarquons une ressemblance visible à l'oeil nu entre certaines maladies qui concernent la

classe ”tomato”, nous pouvons le voir sur la matrice de confusion que nous présentons dans la figure 3.5 où nous constatons une confusion sur plusieurs images entre les classes ”late blight” et ”early blight” , ”septoria leaf spot” et ”leaf mold”, ou encore entre ”mites two spotted spider mite” et ”target spot”. La figure 3.7 montre quelques exemples de ces classes.
Les taux de confusion au niveau de la deuxième méthode sont faibles et se justifient très souvent par des pathologies qui partagent des propriétés similaires. C'est le cas des deux classes ”potato late blight” et ”tomato late blight”. La figure 3.8 montre quelques exemples de ces confusion. En effet, le modèle proposé arrive à identifier la maladie ”late blight” mais confond entre les deux types de plante.

- Le deuxième protocole s'est avéré plus simple en comparaison avec le premier protocole, il permet un temps d'exécution plus court ainsi qu'une facilité d'implémentation du modèle pour une application web en offrant des performances satisfaisantes comme le montre le digramme de la figure 3.6.

- En comparant les résultats obtenus avec ceux présentées dans l'état de l'art et plus précisément dans le tableau 1.1, au niveau de l'aliment Tomato, on remarque qu'avec notre modèle on a pu dépasser les auteurs de [37] qui ont utilisé l'architecture AlexNet, avec 1,2% de précision. D'autre part, avec le modèle ResNet152v2 on dépasse la précision obtenue par les auteurs de [38] (avec 99,18%) avec l'architecture GoogleNet dans le memê aliment Tomato comme indique la figure 3.9. Pour les autres aliments, il n'y a pas encore de références.

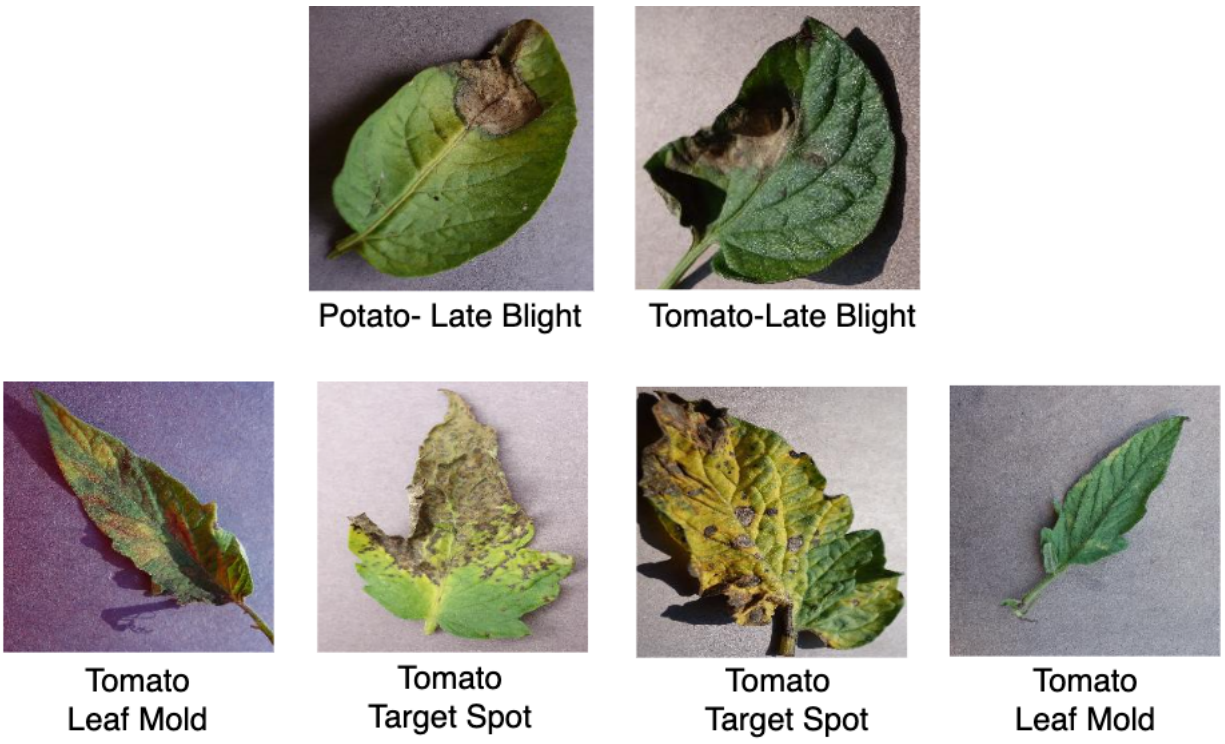


Fig. 3.7 : Exemples des images mal classées

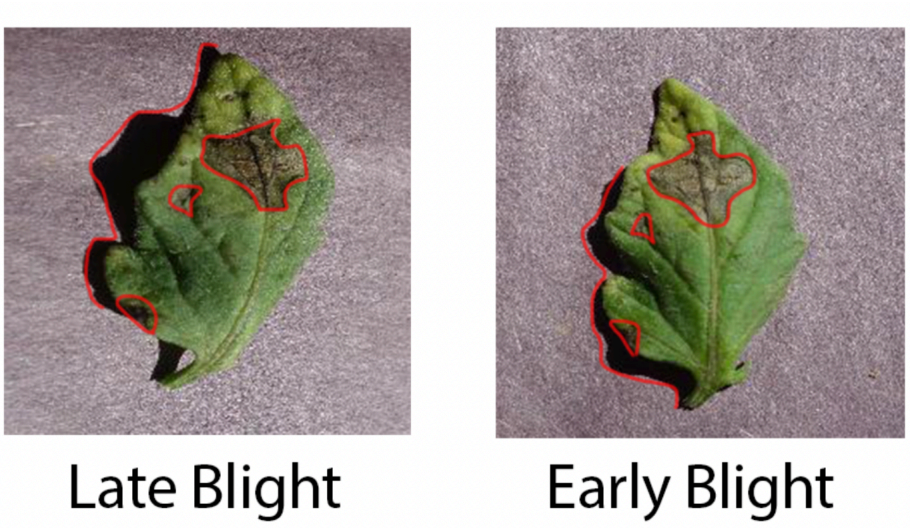


FIG. 3.8 : Comparaison entre les deux pathologies (early blight et late blight) de Tomato

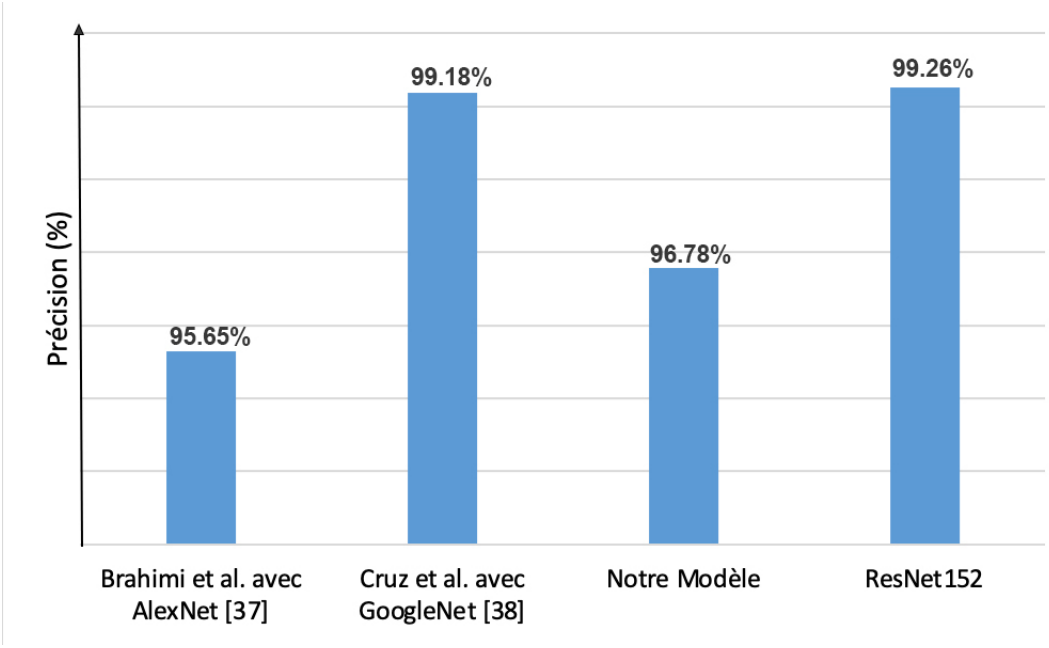


FIG. 3.9 : Comparaison des résultats obtenus avec ceux de l'état de l'art (Tomato)

3.8 Conclusion

Dans ce chapitre nous avons présenté les tests établis pour évaluer le système de de diagnostic des maladies des plantes, fondé sur les réseaux de neurones convolutifs. Notre travail a porté sur l'étude de trois modèles qui ont montré leurs prouesses dans le domaine de vision par ordinateur à savoir ResNet152, AlexNet, VGG16. D'après les expériences et tests menés, nous pouvons dire que l'utilisation réussie des CNN requiert des moyens de calcul robustes, et consomme de l'espace mémoire. Par conséquent, comme notre objectif final est de développer une application web, nous avons proposé une architecture CNN

adaptée qui répond aux exigences du web tout en offrant de bonnes performances.

Dans le chapitre suivant, nous procédons à l'implémentation de notre modèle sur une application web, dans le but de ressortir avec un système prêt à être mis en place pour réaliser le diagnostic des maladies des plantes.

Chapitre 4

Implémentation de la solution

4.1 Introduction

Afin de commencer à utiliser un modèle pour la prise de décision pratique, il doit être déployé efficacement en production. Si on peut pas obtenir de manière fiable des informations pratiques à partir de notre modèle, l'impact de ce dernier est fortement limité.

Le déploiement est la méthode par laquelle on intègre un modèle d'apprentissage automatique dans un environnement de production existant pour prendre des décisions commerciales pratiques basées sur certaines données. C'est l'une des dernières étapes du cycle de vie de l'apprentissage automatique et peut être l'une des plus lourdes.

Le déploiement du modèle nécessite une coordination entre les spécialistes des données, les équipes informatiques, les développeurs de logiciels et les professionnels de l'entreprise pour s'assurer que le modèle fonctionne de manière fiable dans l'environnement de production de l'organisation.

L'objectif de ce travail est de proposer une application web pour le diagnostic des maladies des plantes basée sur un modèle d'apprentissage profond. Ainsi dans ce chapitre nous abordons, les détails de l'implémentation de notre modèle développé précédemment dans une interface web pour pouvoir l'utiliser efficacement, afin de faire le diagnostic des maladies des plantes d'une manière plus simple et le rendre plus accessible aux utilisateurs. Pour l'implémentation nous avons choisi le protocole 02 car plus rapide et plus simple.

4.2 Librairies et Frameworks Utilisés

Pour pouvoir intégrer le modèle à une application web, il existe de nombreux frameworks populaires qui peuvent être utilisés pour effectuer cette tâche, comme Flask, Django et FastAPI [62]. Django est généralement utilisé pour les applications à grande échelle et prend beaucoup de temps à mettre en place, tandis que Flask et Fast API sont généralement utilisés pour déployer rapidement le modèle sur une application web.
Dans ce travail, nous avons choisi les infrastructures logicielles (Framework) suivantes :

4.2.1 FastAPI

FastAPI [63], est un framework web moderne et performant pour construire des APIs avec Python basées sur des indices de type standard. Il possède les caractéristiques clés suivantes :

- **Rapide :** Très haute performance, similaire à NodeJS et Go. L'un des frameworks Python les plus rapides du marché.
- **Rapide à coder :** Il permet d'augmenter considérablement la vitesse de développement.

- **Réduction du nombre de bugs :** Il réduit la possibilité d'erreurs d'origine humaine.
- **Intuitif :** Il offre un excellent support pour l'éditeur, avec des compléments partout et moins de temps de débogage.
- **Simplicité :** Il a été conçu pour être simple à utiliser et à apprendre, de sorte que vous pouvez passer moins de temps à lire la documentation.
- **Robuste :** il fournit un code prêt pour la production avec une documentation interactive automatique.

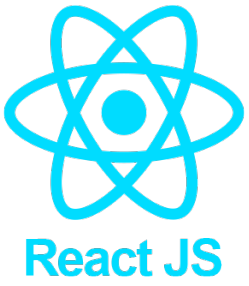
Cette infrastructure logicielle est conçue pour optimiser l'expérience de développement afin qu'on puisse écrire un code simple pour construire des API prêtes à être mises en production.

4.2.2 Uvicorn

Uvicorn est un serveur web compatible ASGI (Async Server Gateway interface) [64]. C'est l'élément de liaison qui gère les connexions web du navigateur ou du client API et permet ensuite à FastAPI de servir la demande réelle.

4.2.3 React JS

React (aussi appelé React.js ou ReactJS) est une bibliothèque JavaScript libre développée par Facebook depuis 2013. Le but principal de cette bibliothèque est de faciliter la création d'application web monopage, via la création de composants dépendant d'un état et générant une page (ou portion) HTML à chaque changement d'état [65].



Cette bibliothèque possède les caractéristiques clés suivantes :

- **Création facile d'applications dynamiques :** React facilite la création d'applications web dynamiques car il nécessite moins de codage et offre plus de fonctionnalités, contrairement à JavaScript, où le codage devient souvent complexe très rapidement.
- **Facile à apprendre :** React est facile à apprendre, car il combine principalement les concepts HTML et JavaScript de base avec quelques ajouts bénéfiques. Néanmoins, comme c'est le cas avec d'autres outils et frameworks, vous devez passer un certain temps pour bien comprendre la bibliothèque de React.
- **Amélioration des performances :** React utilise le "Document Object Model" (DOM) virtuel, créant ainsi des applications web plus rapidement. le DOM Virtuel compare les états précédents des composants et ne met à jour que les éléments

du DOM réel qui ont été modifiés, au lieu de mettre à nouveau à jour tous les composants, comme le font les applications web classiques [66].

- **Des outils dédiés pour un débogage facile :** Facebook a créé une extension Chrome qui peut être utilisée pour débugger les applications React. Cela rend le processus de déboguage des applications web React plus rapide et plus facile.

4.3 Développement

Dans la section suivante, nous allons nous pencher sur les aspects les plus importants de la construction d'une application web. Ce sont les grandes pièces qui sont en général toujours nécessaires. Dans une certaine mesure, ces éléments peuvent être similaires au développement de logiciels mobiles, de jeux ou d'autres types de logiciels. Ce travail se concentre principalement sur les aspects techniques généraux de la création d'une application web monopage. Il convient de mentionner qu'il existe également d'autres aspects importants. Par exemple, la sécurité des applications est un sujet trop vaste pour être abordé en profondeur dans ce travail, mais qui doit être mentionné. C'est une chose à laquelle un développeur doit penser tout en développant les fonctionnalités backend et frontend. C'est également une partie importante de l'infrastructure et de la surveillance de l'application.

Le développement de notre application web pour le diagnostic des maladies des plantes, se fait en deux partie :

4.3.1 Coté serveur (Backend)

Le backend est le côté serveur du site web. Il stocke principalement notre modèle d'apprentissage profond qu'on a développé précédemment ainsi que les noms des classes et les pré-traitements faits sur les images, et s'assure également que tout ce qui se trouve du côté client du site web fonctionne correctement. C'est la partie du site web qu'on ne peut pas voir et avec laquelle on ne peut pas interagir. C'est la partie du logiciel qui n'est pas en contact direct avec les utilisateurs. Les parties et les caractéristiques développées dans le backend sont indirectement accessibles aux utilisateurs par le biais d'une application frontend. Les activités telles que l'écriture d'API, la création de bibliothèques et le travail avec des composants du système sans interface utilisateur ou même des systèmes de programmation scientifique, font également partie du backend.

Pour le développement du backend de notre application nous avons choisi la bibliothèque FastAPI de python comme mentionné précédemment. ce dernier passe par plusieurs étapes comme suit :

Installer les pré-requis nécessaires

Avant de commencer la partie codage, nous devons télécharger FastAPI et d'autres bibliothèques. Ici, nous utilisons un environnement virtuel, où toutes les bibliothèques sont gérées, ce qui facilite le travail de développement et de déploiement.

- **Installation de FastAPI**
  L'installation de FastAPI est la même que celle de tout autre module python, mais ce dernier n'a pas de serveur de développement intégré. Donc, nous allons utiliser Uvicorn, un serveur ASGI (Asynchronous Server Gateway Interface), pour servir FastAPI.
  L'installation des deux modules se fait en utilisant la commande suivante :

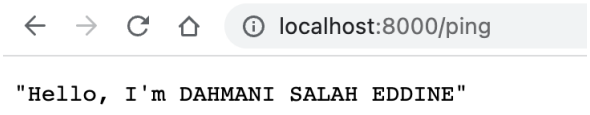
  ```
  pip install fastapi uvicorn
  ```

  Ensuite, nous créons notre répertoire, qui va contenir tous les fichiers nécessaires pour notre application web coté serveur. Par la suite, nous créons le fichier principale "main.py" qui gère toutes les fonctionnalités de notre application web.

- **Test de notre API** : Le fichier **"main.py"** a défini toutes les opérations du chemin, pour exécuter ce fichier, il faut ouvrir le terminal dans notre répertoire et écrire la commande suivante :

  ```
  uvicorn main:app --reload
  ```

  Après avoir exécuté cette commande on aura la fenêtre ci-dessous :



- **Importer les différentes librairies**
  Afin que notre programme puisse fonctionner correctement, à priori, il faut importer toutes les librairies utilisées lors du développement du notre modèle CNN, pour ce faire, on génère un fichier qui s'appelle **"requirments.txt"** a travers la commande :

  ```
  pip install pipreqs
  pipreqs Web_application/api
  ```

  Après l'exécution des deux commandes précédentes on aura notre fichier "requirmenets.txt" comme suit :

  On installe les librairies en utilisant la commande :

  ```
  pip install -r requirements.txt
  ```

Maintenant que notre environnement de développement est prêt, on aborde la partie programmation du côté serveur.

Pré-traitement des images

Avant l'exécution de la tâche de détection des maladies, et après avoir chargé une image dans l'application, un pré-traitement a été appliqué sur ces images pour que notre modèle puisse les traiter correctement. Premièrement on utilise la fonction **"numpy.array()"** qui prend une image comme argument et la convertit en tableau Numpy. Ensuite, à l'aide de la fonction **"numpy.expend_dims"**, on ajoute une dimension d'une unité à notre image qui est représentée par un tableau de valeur dans l'intervalle [0, 255], finalement, on fait un redimensionnement de ces valeurs dans l'intervalle [0, 1] pour avoir des images avec les mêmes caractéristiques avec lesquelles notre CNN a été entraîné comme indique le code ci-dessous :

```
def read_file_as_image(data) -> np.ndarray:
    image1 = np.array(Image.open(BytesIO(data)))
    resized_image = cv2.resize(image1, (256,256))
    img_batch = np.expand_dims(resized_image, 0)
    img = img_batch/255
    return img
```

Diagnostic

C'est la que vient l'étape la plus importante de notre backend, c'est la partie du code qui se charge de faire le diagnostic des différentes maladies.
Lorsque l'utilisateur charge une image d'une feuille de plante, l'application web doit détecter le type de plante en question ainsi que la maladie associé à ce type. Pour cela, nous avons besoin du fichier modèle que nous avons enregistré précédemment dans le même dossier de projet. Pour le chargement du fichier de notre modèle, on utilise la fonction load_model comme suit :

```
MODEL = tf.keras.models.load_model("saved_models/all_f_Scratch_final.h5")
```

Maintenant, nous avons un autre élément "@app.route ('/predict')", celui-ci fait correspondre la fonction **"predict()"** avec *l'URL /predict*, cette dernière, comme son nom l'indique, prend l'image donnée par l'utilisateur, effectue tous les pré-traitements, et la fait passer par les différentes couches du modèle CNN proposé, donnant finalement un score de confiance sur l'appartenance de cette image à l'une des classes prédéfinies en utilisant la fonction **softmax()**. Enfin, nous obtenons le type d'aliment ainsi que la maladie associée, en utilisant **argmax()** sur le score de confiance du modèle. On l'utilisera comme index pour rechercher dans le tableau "class_names" qui contient les différentes classes d'aliments que nous avons. Enfin, on multiplie le score de confiance du modèle par 100 afin que la plage de score soit comprise entre 1 et 100. Les figures 4.1, 4.2 montrent l'interface du notre côté serveur (Backend) :

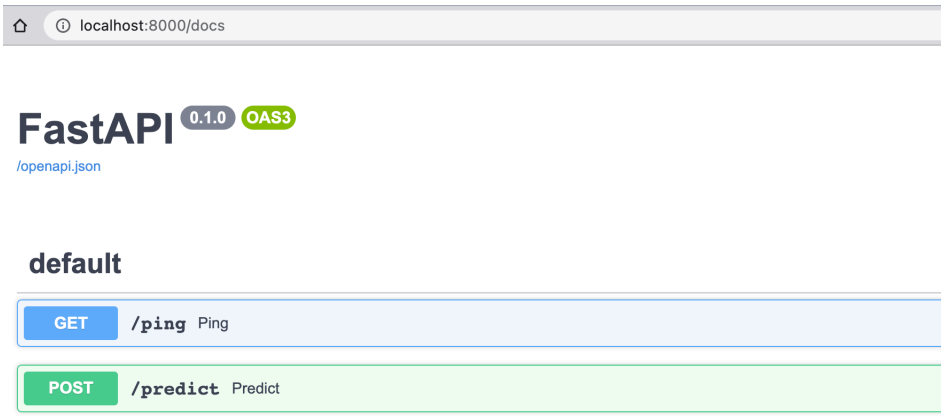


FIG. 4.1 : Interface Backend

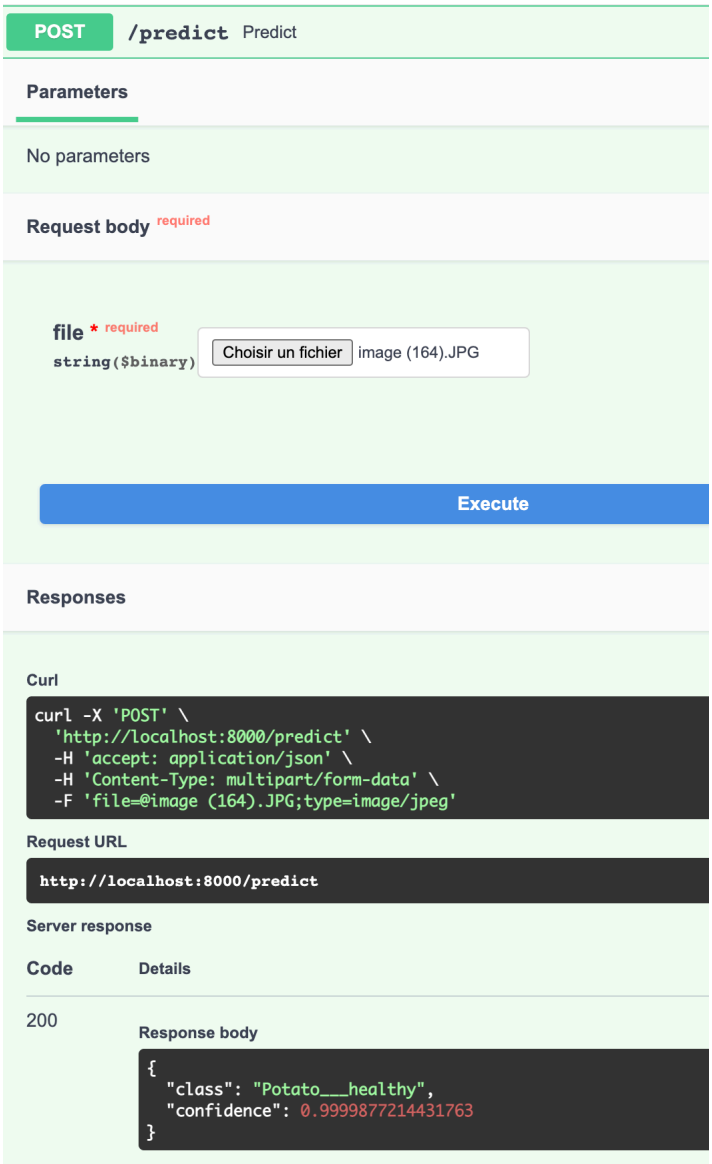


FIG. 4.2 : Interface Backend après exécution de la fonction predict()

4.3.2 Côté utilisateur (Frontend)

La partie d'un site web avec laquelle l'utilisateur interagit directement est appelée frontend. On l'appelle aussi le "côté client" de l'application. Elle comprend tout ce que les utilisateurs voient dans le navigateur : couleurs et styles de texte, images, graphiques et tableaux, boutons, couleurs et menu de navigation. HTML, CSS et JavaScript sont les langages utilisés pour le développement du frontend. La structure, la conception, le comportement et le contenu de tout ce que l'on voit sur les écrans des navigateurs lorsque les sites web, les applications web ou les applications mobiles sont ouverts, sont mis en œuvre par les développeurs de frontend. La réactivité et la performance sont deux objectifs principaux du frontend.

Nous devons nous assurer que le site est réactif, c'est-à- dire une interface homme machine doit être ergonomique aussi bien qu'efficace. De plus, ces interfaces doivent être faciles à utiliser et compréhensibles par les utilisateurs.
Comme mentionné précédemment on va utiliser la bibliothèque React Js, HTML et Css pour la conception de notre interface web, avant de commencer la partie développement on doit installer le logiciel **Node.js**.

Répertoires

L'application React crée automatiquement les dossiers requis, comme indiqué ci-dessous dans la figure 4.3 :

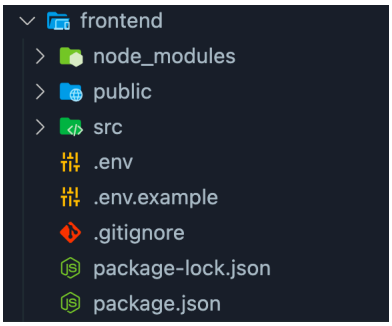


FIG. 4.3 : Répertoire de la partie Frontend

- **Répertoire node modules :** ce dossier contient toutes les dépendances et sous-dépendances de notre projet. Nous avons seulement React, et React scripts mais les scripts React ont beaucoup d'autres dépendances qui sont présentes dans ce dossier.

- **Répertoire public :** Fondamentalement, ce dossier contient 3 fichiers comme suit :
  - icon.png : c'est un fichier d'icône qui contient une icône qui s'affiche sur le navigateur.
  - index.html : c'est un fichier important. Grâce à ce fichier, le dossier public, connu sous le nom de "dossier racine", est servi par le serveur web à la fin. Index.html est une page HTML unique présente dans ce projet.

- **Répertoire src :**
  Ce dossier contient le code source actuel du développement. C'est l'endroit où notre

application React est présente. Nous pouvons créer notre propre sous-répertoire à l'intérieur de ce répertoire. Pour des reconstructions plus rapides, les fichiers à l'intérieur de ce seul dossier sont traités par "Webpack". Cependant, ce dossier contient déjà les fichiers suivants :
- Home.js : c'est le fichier principal qui contient le design de notre application ainsi que tous les détails graphiques de l'interface (taille, couleur, emplacements,...etc.).
- App.css : ce fichier contient des classes CSS ou, en d'autres termes, un style qui est utilisé par le fichier App.js.
- App.js : est un exemple de composant React appelé "App" que nous obtenons gratuitement lorsque nous créons une nouvelle application. C'est un composant qui sera chargé sous le fichier index.js. Si nous effectuons un changement dans le composant HTML du fichier app.js et le sauvegardons, il sera reflété dans localhost ://3000.
- App.test.js : c'est le fichier de test qui nous permet de créer des tests unitaires pour différentes unités ou pour différents composants.
- index.css : il stocke le style de base de notre application.
- index.js : c'est le fichier qui sera appelé une fois que nous aurons lancé le projet.

- **Le fichier .gitignore :** Ce fichier est utilisé par l'outil de contrôle de la source pour identifier les fichiers et les dossiers qui doivent être inclus ou ignorés lors de la transmission du code.

- **Le fichier package.json :** C'est un fichier standard que l'on trouve dans chaque projet. Il contient des informations comme le nom du projet, les versions, les dépendances, etc. Chaque fois que nous installons une bibliothèque tierce, elle est automatiquement enregistrée dans ce fichier.

- **Le fichier .env :** c'est le fichier où on stocke une clé d'API de notre côté serveur (Backend) développé précédemment, pour que toutes les demandes inconnues de chemins relatifs provenant du front-end seront envoyées au serveur back-end fonctionnant sur le port 8000.
  le fichier .env contient seulement la ligne de code suivante :

  ```
  REACT_APP_API_URL=http://localhost:8000/predict
  ```

4.3.3 Tests

Maintenant que notre application de detection des maladies des plantes est prête à l'utilisation on procède aux tests du bon fonctionnement de cette dernière. Pour ce faire, on doit exécuter au même temps le fichier **main.py** de notre backend ainsi que notre interface utilisateur avec la commande suivante :

```
npm run start
```

Après l'exécution de la commande ci-dessus la fenêtre de notre interface utilisateur s'ouvre automatiquement comme indique la figure 4.4 :

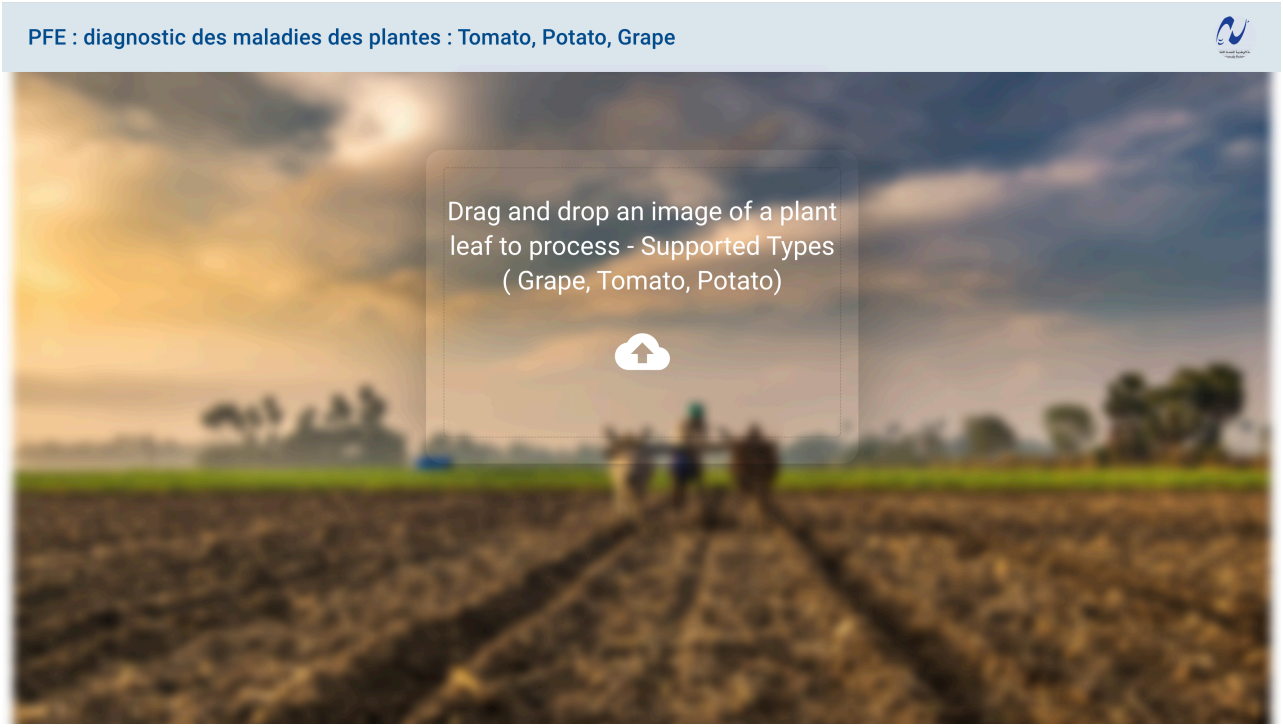


FIG. 4.4 : Page principale de notre interface web

Après avoir chargé une image d'une feuille de plante dans l'application on aura notre diagnostic avec un pourcentage de confiance, selon l'état de la feuille comme indique la figure 4.5 :

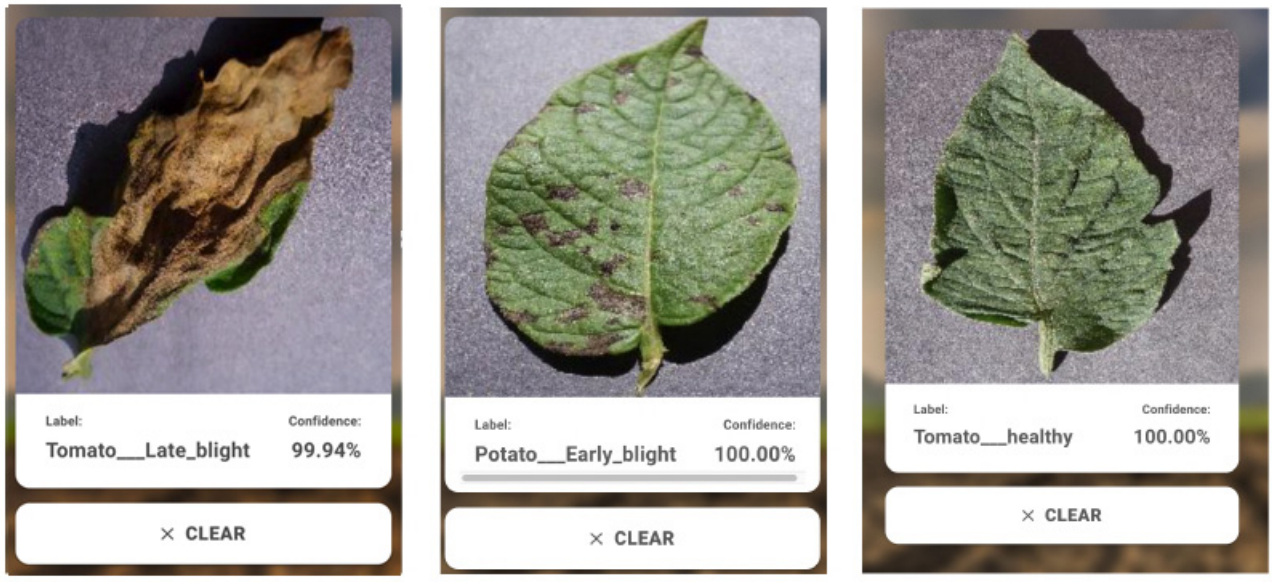


FIG. 4.5 : Réponse de l'application après l'exécution

4.4 Conclusion

Dans ce chapitre nous avons présenté les concepts de base ainsi que les différentes étapes suivies lors du développement de notre interface web. D'autre part, on a expliqué l'utilisation des infrastructures logicielles tels que React Js et FastAPI pour assurer une bonne expérience utilisateur, dans le but de faire le diagnostic des maladies des plantes (Tomato, Grape, Potato), ce à travers l'implémentation de notre modèle que nous avons développé dans le chapitre 03, en utilisant les réseaux de neurones convolutifs.

Conclusion et perspectives

L'objectif de ce travail est de développer une interface web pour le diagnostic et la classification des maladies des plantes, fondé sur les réseaux de neurones convolutifs. Les CNN ont été évalués en comme des systèmes end-to-end qui reçoivent les images des feuilles de plantes à l'entrée et fournissent une décision automatique sur l'état ou la pathologie de la plante. En second lieu, nous avons fait le déploiement du modèle que nous avons proposé dans une interface web pour que ça soit un système efficace et accessible aux utilisateurs.

Les tests ont porté sur une sous-ensemble de trois types de cultures de la base de données PlantVillage en utilisant les modèles AlexNet, VGG16, ResNet152, et un modèle que nous avons proposé. À partir des expériences réalisées, nous concluons que :

- Pour le problème de détection, ResNet152v2 et VGG16 offrent les meilleures performances presque optimales (supérieures à 98%), néanmoins, ces modèles sont gourmands en termes de capacité de calcule et ressources matérielles ainsi qu'en temps d'exécution.
- La classification de plusieurs types de pathologies pouvant affecter un même type de culture a été achevée avec des précisions de l'ordre de 96% au niveau de notre modèle, ce qui confirme la robustesse de notre système d'identification des différentes maladies des plantes.
- L'utilisation efficace d'un CNN est un problème de conception expérimentale, où plusieurs tests sont nécessaires pour trouver la configuration adéquate à l'application à envisager.
- Il est important d'adapter la taille des données de l'entrée à la profondeur du réseau pour obtenir des caractéristiques discriminantes à la sortie du bloc de convolution. Une image très grande ne peut être entraînée par un réseau contenant un petit bloc de convolution (comme AlexNet). En revanche, les caractéristiques d'une petite image seront perdues à la sortie du bloc de convolution, si le réseau est très profond (Cas des autres modèles).
- L'implémentation des modèles d'apprentissage profond dans les applications web demandes beaucoup d'espace pour l'héberger et le rendre accessible au public, car on a besoin de charger toutes les bibliothèques utilisées comme tenserflow et keras sans compter la taille du modèle qui représente un critère important dans le choix de ce dernier.

Enfin, les résultats obtenus peuvent être améliorés encore avec des moyens de calcul adéquats pour effectuer une conception plus efficace des modèles CNN profonds. Ainsi, ce travail peut être étendu à d'autres types de cultures proposés par la base Plant Village ou d'autres bases publiques, pour confirmer la robustesse des systèmes développés. Aussi, il serait très intéressant de tester d'autres modèles CNN, comme Xception. Comme perspectives de recherche possibles pour le travail qui concerne le déploiement, il est très intéressant que ce système soit implémenté dans une application mobile pour faciliter la tâche de traitement aux utilisateurs grâce à la caméra présente dans le smartphone, et sans avoir recours à une connexion internet.

Bibliographie

1. STRANGE, Richard N ; SCOTT, Peter R. Plant disease : a threat to global food security. *Annual review of phytopathology*. 2005, t. 43, n° 1, p. 83-116.
2. EBRAHIMI, MA ; KHOSHTAGHAZA, Mohammad Hadi ; MINAEI, Saeid ; JAMSHIDI, Bahareh. Vision-based pest detection based on SVM classification method. *Computers and Electronics in Agriculture*. 2017, t. 137, p. 52-58.
3. CHENG, Xi ; ZHANG, Youhua ; CHEN, Yiqiong ; WU, Yunzhi ; YUE, Yi. Pest identification via deep residual learning in complex background. *Computers and Electronics in Agriculture*. 2017, t. 141, p. 351-356.
4. EBRAHIMI, MA ; KHOSHTAGHAZA, Mohammad-Hadi ; MINAEI, Saeid ; JAMSHIDI, Bahareh. Vision-based pest detection based on SVM classification method. *Computers and Electronics in Agriculture*. 2017, t. 137, p. 52-58.
5. LI, Lili ; ZHANG, Shujuan ; WANG, Bin. Plant disease detection and classification by deep learning—a review. *IEEE Access*. 2021, t. 9, p. 56683-56698.
7. SANKARAN, Sindhuja ; MISHRA, Ashish ; EHSANI, Reza ; DAVIS, Cristina. A review of advanced techniques for detecting plant diseases. *Computers and electronics in agriculture*. 2010, t. 72, n° 1, p. 1-13.
8. RATHOD, Arti N ; TANAWAL, Bhavesh ; SHAH, Vatsal. Image processing techniques for detection of leaf disease. *International Journal of Advanced Research in Computer Science and Software Engineering*. 2013, t. 3, n° 11.
9. GARRETT, Karen A ; NITA, Mizuho ; DE WOLF, ED ; ESKER, Paul David ; GOMEZ-MONTANO, Lorena ; SPARKS, Adam H. Plant pathogens as indicators of climate change. In : *Climate change*. Elsevier, 2016, p. 325-338.
10. LUCAS, George B ; CAMPBELL, C Lee ; LUCAS, Leon T. *Introduction to plant diseases : identification and management*. Springer Science & Business Media, 1992.
11. SONKA, Milan ; HLAVAC, Vaclav ; BOYLE, Roger. Image pre-processing. In : *Image Processing, Analysis and Machine Vision*. Boston, MA : Springer US, 1993, p. 56-111. ISBN 978-1-4899-3216-7. Disp. à l'adr. DOI : 10.1007/978-1-4899-3216-7_4.
12. BERA, Tanmoy ; DAS, Ankur ; SIL, Jaya ; DAS, Asit K. A survey on rice plant disease identification using image processing and data mining techniques. In : *Emerging Technologies in Data Mining and Information Security*. Springer, 2019, p. 365-376.
13. VINCENT, Guesdon. *Détection efficace de contours d'images*. 2004. Thèse de doct. Université du Québec en Outaouais.

14. SINGH, Vijai ; MISRA, Ak K. Detection of plant leaf diseases using image segmentation and soft computing techniques. *Information processing in Agriculture*. 2017, t. 4, n° 1, p. 41-49.

15. KUMAR, Gaurav ; BHATIA, Pradeep Kumar. A detailed review of feature extraction in image processing systems. In : *2014 Fourth international conference on advanced computing & communication technologies*. 2014, p. 5-12.

16. VISHNOI, Vibhor Kumar ; KUMAR, Krishan ; KUMAR, Brajesh. Plant disease detection using computational intelligence and image processing. *Journal of Plant Diseases and Protection*. 2021, t. 128, n° 1, p. 19-53.

17. WAGHMARE, Harshal ; KOKARE, Radha ; DANDAWATE, Yogesh. Detection and classification of diseases of grape plant using opposite colour local binary pattern feature and machine learning for automated decision support system. In : *2016 3rd international conference on signal processing and integrated networks (SPIN)*. 2016, p. 513-518.

18. PIRES, Rillian Diello Lucas ; GONÇALVES, Diogo Nunes ; ORUÊ, Jonatan Patrick Margarido ; KANASHIRO, Wesley Eiji Sanches ; RODRIGUES JR, Jose F ; MACHADO, Bruno Brandoli ; GONÇALVES, Wesley Nunes. Local descriptors for soybean disease recognition. *Computers and Electronics in Agriculture*. 2016, t. 125, p. 48-55.

19. DANDAWATE, Yogesh ; KOKARE, Radha. An automated approach for classification of plant diseases towards development of futuristic Decision Support System in Indian perspective. In : *2015 International conference on advances in computing, communications and informatics (ICACCI)*. 2015, p. 794-799.

20. PYDIPATI, R ; BURKS, TF ; LEE, WS. Identification of citrus disease using color texture features and discriminant analysis. *Computers and electronics in agriculture*. 2006, t. 52, n° 1-2, p. 49-59.

21. DESHAPANDE, Anupama S ; GIRADDI, Shantala G ; KARIBASAPPA, KG ; DESAI, Shrinivas D. Fungal disease detection in maize leaves using haar wavelet features. In : *Information and Communication Technology for Intelligent Systems*. Springer, 2019, p. 275-286.

22. AHMAD, Nisar ; ASIF, Hafiz Muhammad Shahzad ; SALEEM, Gulshan ; YOUNUS, Muhammad Usman ; ANWAR, Sadia ; ANJUM, Muhammad Rizwan. Leaf image-based plant disease identification using color and texture features. *Wireless Personal Communications*. 2021, t. 121, n° 2, p. 1139-1168.

23. HUGHES, David ; SALATHÉ, Marcel et al. An open access repository of images on plant health to enable the development of mobile disease diagnostics. *arXiv preprint arXiv :1511.08060*. 2015.

24. KAWASAKI, Yusuke ; UGA, Hiroyuki ; KAGIWADA, Satoshi ; IYATOMI, Hitoshi. Basic study of automated diagnosis of viral plant diseases using convolutional neural networks. In : *International symposium on visual computing*. 2015, p. 638-645.

25. FUJITA, Erika ; KAWASAKI, Yusuke ; UGA, Hiroyuki ; KAGIWADA, Satoshi ; IYATOMI, Hitoshi. Basic investigation on a robust and practical plant diagnostic system. In : *2016 15th IEEE international conference on machine learning and applications (ICMLA)*. 2016, p. 989-992.

26. SLADOJEVIC, Srdjan ; ARSENOVIC, Marko ; ANDERLA, Andras ; CULIBRK, Dubravko ; STEFANOVIC, Darko. Deep neural networks based recognition of plant diseases by leaf image classification. *Computational intelligence and neuroscience*. 2016, t. 2016.

27. MOHANTY, Sharada P ; HUGHES, David P ; SALATHÉ, Marcel. Using deep learning for image-based plant disease detection. *Frontiers in plant science*. 2016, t. 7, p. 1419.

28. AMARA, Jihen ; BOUAZIZ, Bassem ; ALGERGAWY, Alsayed. A deep learning-based approach for banana leaf diseases classification. *Datenbanksysteme für Business, Technologie und Web (BTW 2017)-Workshopband*. 2017.

29. LIU, Weibo ; WANG, Zidong ; LIU, Xiaohui ; ZENG, Nianyin ; LIU, Yurong ; ALSAADI, Fuad E. A survey of deep neural network architectures and their applications. *Neurocomputing*. 2017, t. 234, p. 11-26.

30. ZHANG, Xihai ; QIAO, Yue ; MENG, Fanfeng ; FAN, Chengguo ; ZHANG, Mingming. Identification of maize leaf diseases using improved deep convolutional neural networks. *Ieee Access*. 2018, t. 6, p. 30370-30377.

31. TOO, Edna Chebet ; YUJIAN, Li ; NJUKI, Sam ; YINGCHUN, Liu. A comparative study of fine-tuning deep learning models for plant disease identification. *Computers and Electronics in Agriculture*. 2019, t. 161, p. 272-279.

32. RANGARAJAN, Aravind Krishnaswamy ; PURUSHOTHAMAN, Raja ; RAMESH, Aniirudh. Tomato crop disease classification using pre-trained deep learning algorithm. *Procedia computer science*. 2018, t. 133, p. 1040-1047.

33. MA, Juncheng ; DU, Keming ; ZHENG, Feixiang ; ZHANG, Lingxian ; GONG, Zhihong ; SUN, Zhongfu. A recognition method for cucumber diseases using leaf symptom images based on deep convolutional neural network. *Computers and electronics in agriculture*. 2018, t. 154, p. 18-24.

34. SARDOGAN, Melike ; TUNCER, Adem ; OZEN, Yunus. Plant leaf disease detection and classification based on CNN with LVQ algorithm. In : *2018 3rd International Conference on Computer Science and Engineering (UBMK)*. 2018, p. 382-385.

35. WANG, Guan ; SUN, Yu ; WANG, Jianxin. Automatic image-based plant disease severity estimation using deep learning. *Computational intelligence and neuroscience*. 2017, t. 2017.

36. DURMUŞ, Halil ; GÜNEŞ, Ece Olcay ; KIRCI, Mürvet. Disease detection on the leaves of the tomato plants by using deep learning. In : *2017 6th International Conference on Agro-Geoinformatics*. 2017, p. 1-5.

37. BRAHIMI, Mohammed ; BOUKHALFA, Kamel ; MOUSSAOUI, Abdelouahab. Deep learning for tomato diseases : classification and symptoms visualization. *Applied Artificial Intelligence*. 2017, t. 31, n° 4, p. 299-315.

38. CRUZ, Albert C ; LUVISI, Andrea ; DE BELLIS, Luigi ; AMPATZIDIS, Yiannis. Vision-based plant disease detection system using transfer and deep learning. In : *2017 asabe annual international meeting*. 2017, p. 1.

39. LOEY, Mohamed ; ELSAWY, Ahmed ; AFIFY, Mohamed. Deep learning in plant diseases detection for agricultural crops : a survey. *International Journal of Service Science, Management, Engineering, and Technology (IJSSMET)*. 2020, t. 11, n° 2, p. 41-58.

40. GANDHI, Rutu ; NIMBALKAR, Shubham ; YELAMANCHILI, Nandita ; PONKSHE, Surabhi. Plant disease detection using CNNs and GANs as an augmentative approach. In : *2018 IEEE International Conference on Innovative Research and Development (ICIRD)*. 2018, p. 1-5.

41. ZOUINAR, Moustafa. Évolutions de l'Intelligence Artificielle : quels enjeux pour l'activité humaine et la relation Humain-Machine au travail ? *Activités*. 2020, n° 17-1.

42. CIRESAN, Dan ; GIUSTI, Alessandro ; GAMBARDELLA, Luca ; SCHMIDHUBER, Jürgen. Deep neural networks segment neuronal membranes in electron microscopy images. *Advances in neural information processing systems*. 2012, t. 25.

43. LIU, Xiaolong ; DENG, Zhidong ; YANG, Yuhan. Recent progress in semantic image segmentation. *Artificial Intelligence Review*. 2019, t. 52, n° 2, p. 1089-1106.

44. BOUIBED, M. L. Writer Retrieval In Document Images. *Thèse de Doctorat LMD en Télécommunications,FEI,USTHB*. 2020, t. 152.

45. ROSENBLATT, Frank. The perceptron : a probabilistic model for information storage and organization in the brain. *Psychological review*. 1958, t. 65, n° 6, p. 386.

47. DECARO, Cristoforo ; MONTANARI, Giovanni Battista ; MOLINARI, Riccardo ; GILBERTI, Alessio ; BAGNOLI, Davide ; BIANCONI, Marco ; BELLANCA, Gaetano. Machine learning approach for prediction of hematic parameters in hemodialysis patients. *IEEE Journal of Translational Engineering in Health and Medicine*. 2019, t. 7, p. 1-8.

48. RUMELHART, David E ; HINTON, Geoffrey E ; WILLIAMS, Ronald J. Learning representations by back-propagating errors. *nature*. 1986, t. 323, n° 6088, p. 533-536.

49. HUBEL, David H ; WIESEL, Torsten N. Receptive fields, binocular interaction and functional architecture in the cat's visual cortex. *The Journal of physiology*. 1962, t. 160, n° 1, p. 106.

50. FUKUSHIMA, Kunihiko ; MIYAKE, Sei. Neocognitron : A self-organizing neural network model for a mechanism of visual pattern recognition. In : *Competition and cooperation in neural nets*. Springer, 1982, p. 267-285.

51. LECUN, Yann ; JACKEL, Larry ; BOTTOU, Leon ; BRUNOT, A ; CORTES, Corinna ; DENKER, John ; DRUCKER, Harris ; GUYON, Isabelle ; MULLER, UA ; SACKINGER, Eduard et al. Comparison of learning algorithms for handwritten digit recognition. In : *International conference on artificial neural networks*. 1995, t. 60, p. 53-60.

52. LECUN, Yann ; BENGIO, Yoshua ; HINTON, Geoffrey. Deep learning. *nature*. 2015, t. 521, n° 7553, p. 436-444.

53. BOROVKOV, Andriy. *Image Classification with Deep Learning*. 2017. Thèse de doct. Universität Hamburg.

54. KRIZHEVSKY, Alex ; SUTSKEVER, Ilya ; HINTON, Geoffrey E. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*. 2012, t. 25.

57. DENG, Jia ; DONG, Wei ; SOCHER, Richard ; LI, Li-Jia ; LI, Kai ; FEI-FEI, Li. Imagenet : A large-scale hierarchical image database. In : [s. d.].

Webographie

6. *PlantVillage Dataset | Kaggle* [https://bit.ly/3m2HYNV]. [s. d.].
46. *Modèles du neurone biologique — Wikipédia* [https://bit.ly/3PltGpc]. [s. d.].
55. *DataScientest.com | Formations en Data Science* [https://bit.ly/3NiLYGO]. [s. d.].
56. *Introduction to Residual Networks - GeeksforGeeks* [https://bit.ly/3m4u6mx]. [s. d.].
58. *ML | Introduction to Transfer Learning - GeeksforGeeks* [https://bit.ly/3lWoniH]. [s. d.].
59. *What is Python ? Executive Summary | Python.org* [https://bit.ly/3ze7tUw]. [s. d.].
60. *Scikit-learn — Wikipédia* [https://bit.ly/3xat25Q]. [s. d.].
61. *Paperspace Gradient : A Modern PaaS for Machine Learning – The New Stack* [https://bit.ly/3x2v084]. [s. d.].
62. *Deploying ML Models as API using FastAPI - GeeksforGeeks* [https://bit.ly/3yhJWRG]. [s. d.].
63. *FastAPI* [https://bit.ly/3nii6hS]. [s. d.].
64. *Uvicorn* [https://bit.ly/3xUeHL1]. [s. d.].
65. *React – Une bibliothèque JavaScript pour créer des interfaces utilisateurs* [https://bit.ly/3A4vVYP]. [s. d.].
66. *The Best Guide to Know What Is React [Updated]* [https://bit.ly/3OFvSGR]. [s. d.].