Université Abou Bekr Belkaid
Tlemcen Algérie
جامعة أبي بكر بلقايد
تلمسان الجزائر

République Algérienne Démocratique et Populaire

Université Abou Bekr Belkaid– Tlemcen

Faculté des Sciences

Département d'Informatique

Mémoire de fin d'études

pour l'obtention du diplôme de Master en Informatique

Option: *Modèles Intelligents et Décision (M.I.D)*

Thème

Nabet-ID : Application Mobile pour la Reconnaissance des Plantes par Deep Learning

Réalisé par :

- **KHITRI Hanane**

Présenté le 03/07/2022 devant le jury composé de MM.

- *M. BENAZZOUZ MOURTADA* *(Président)*
- *M. ABDERRAHIM ALAEDDINE* *(Examinateur)*
- *M. ZIANI-CHERIF SALIM* *(Encadreur)*

Année universitaire : 2021-2022

REMERCIEMENTS

Qu'il me soit permis, **M. ZIANI-CHERIF Salim**, de vous exprimer mes remerciements, ma gratitude, et tout le respect pour votre assistance continue.

J'exprime aussi ma gratitude à messieurs **BENAZZOUZ Mourtada** et **ABDERRAHIM Alaeddine** qui ont eu l'amabilité d'évaluer ce travail.

J’adresse mes sincères remerciements à tous les professeurs, et toutes les personnes qui par leurs paroles, leurs écrits, leurs conseils et leurs critiques ont guidé mes réflexions.

Table des matières

Introduction Générale **9**

Chapitre 1 : Application de reconnaissance de plantes **11**

1.1 Introduction 11

1.2 Présentation des applications les plus connues 11

1.2.1 LeafSnap 11

1.2.2 Pl@ntnet 12

1.2.3 Flora Incognita 13

1.2.4 PlantSnap 14

1.2.5 Comparaison entre les applications 14

1.3 Pourquoi les applications de reconnaissance de plantes utilisent-elles le deep learning 15

1.4 Etudes et publications scientifiques 16

1.5 Conclusion 18

Chapitre 2 : Les réseaux de neurones convolutifs **19**

2.1 Introduction 19

2.2 Les réseaux de neurones convolutifs 19

2.2.1 Les couches de convolution 20

2.2.2 Les couches d'activation (activation layers) 22

2.2.3 Les couches de Pooling (mise en commun) 24

2.2.4 Les couches entièrement connectées 25

2.3 Entraînement d'un réseau 27

2.3.1 La fonction de perte 27

2.3.2 Apprentissage par descente de gradient 28

a. Choix du taux d'apprentissage (α) 29

b. La méthode de descente de gradient stochastique 30

c. La méthode Adam 30

2.3.3 Époques, lots et itérations 30

2.3.4 Sous-apprentissage, sur-apprentissage et régularisation 30

2.4 Architectures des réseaux de neurones convolutifs 33

2.4.1 RNC classiques 33

a.LeNet-5 33
b.AlexNet 33
c.VGG 34
2.4.2 RNC modernes 35
a.Resnet 35
b.Inception 36
c.MobileNet 37
d.MobileNetV2 39
2.5 Conclusion 39
Chapitre 3 : Implémentation **41**
4.1 Introduction 41
4.2 Matériel, langage et outils 41
4.3 Exécution, Résultats 44
4.3.1 Fonction de compilation 44
4.3.2 Bases de données 45
a. Pl@ntent-300k 45
b. Tropic20 46
c. AgrilPlant 47
4.3.3 Techniques d'apprentissage 47
a. Apprentissage par transfert 47
b. Réglage fin 47
c. Augmentation de données 48
4.3.4 La matrice de confusion 49
4.3.5 Traitement d'image 49
4.3.6 Résumé des résultats 49
a. Expérimentation 1 : Pl@ntnet-300k 49
b. Expérimentation 2 : L'apprentissage par transfert 51
c. Expérimentation 3 : Le réglage fin 53
d. Expérimentation 4 : L'augmentation de données 56
e. Expérimentation 5 : Le réglage fin et l'augmentation de données 59
f. Discussion et analyse des résultats 62
4.4 Résultats de prédiction 62

4.5 Déploiement de l'application 65

4.6 Conclusion 67

Conclusion Générale et Perspectives 68

Annexe 69

Bibliographie 70

Table des figures

Figure 1.1 : L'application LeafSnap 12
Figure 1.2 : L'application Pl@ntnet 13
Figure 1.3 : Flora Incognita 13
Figure 1.4 : L'application PlantSnap 14
Figure 1.5 : L'application medicinal plant recognizer [6] 17
Figure 1.6 : Leaf Classifier [7] 18
Figure 2.1 : Apprentissage automatique Vs Apprentissage profond [1] 19
Figure 2.2 : réseau de neurones convolutif [1] 20
Figure 2.3 : Opération de convolution 20
Figure 2.4 : La fonction tanh [9] 22
Figure 2.5 : La fonction ReLU [9] 23
Figure 2.6 : Opération de pooling 24
Figure 2.7 : La fonction Sigmoïde [9] 25
Figure 2.8 : Softmax [9] 26
Figure 2.9 : La fonction de perte en fonction du taux d'apprentissage [9] 28
Figure 2.10 : Courbe de validation et d'entraînement pour la fonction de perte [9] 30
Figure 2.11 : Dropout [11] 31
Figure 2.12 : L'architecture leNet-5 [12] 32
Figure 2.13 : Architecture AlexNet [13] 33
Figure 2.14 : Architecture VGG [14] 34
Figure 2.15 : Réseau simple Vs Resnet [15] 35
Figure 2.16 : Bloc résiduel [15] 35
Figure 2.17 : Convolution 1 par 1 [17] 36
Figure 2.18 : Convolution séparable en fonction de la profondeur [18] 37
Figure 2.19 : Le résidu inversé [19] 38
Figure 3.1 : Anaconda navigator 40
Figure 3.2 : Azure Machine Learning 41
Figure 3.3 : Conversion à Tensorflow Lite 42
Figure 3.4 : L'environnement Android Studio 43
Figure 3.5 : échantillon de la base de données Pl@ntnet300k 45
Figure 3.6 : échantillon de la base de données Tropic20 46
Figure 3.7 : échantillon de la base de données AgrilPlant 46
Figure 3.8 : Réglage fin [9] 48
Figure 3.9 : Courbes d'apprentissage - apprentissage par transfert - Tropic20 50
Figure 3.10 : Courbes d'apprentissage - apprentissage par transfert - AgrilPlant 51
Figure 3.11 : matrice de confusion - apprentissage par transfert - Tropic20 51
Figure 3.12 : matrice de confusion - apprentissage par transfert - AgrilPlant 52
Figure 3.13 : Courbes d'apprentissage - Réglage fin - Tropic20 53
Figure 3.14 : Courbes d'apprentissage - Réglage fin - AgrilPlant 53
Figure 3.15 : Matrice de confusion - Réglage fin - Tropic20 54
Figure 3.16 : Matrice de confusion - Réglage fin - AgrilPlant 55
Figure 3.17 : Courbes d'apprentissage -augmentation de données - Tropic20 56
Figure 3.18 : Courbes d'apprentissage -augmentation de données - -AgrilPlant 56
Figure 3.19 : Matrice de confusion - augmentation de données - Tropic20 57

Figure 3.20 : Matrice de confusion - augmentation de données - AgrilPlant 58
Figure 3.21 : Courbes d'apprentissage - réglage fin et augmentation de données - Tropic20 59
Figure 3.22 : Courbes d'apprentissage - réglage fin et augmentation de données - AgrilPlant 59
Figure 3.23 : Matrice de confusion - Augmentation de données et réglage fin - Tropic20 60
Figure 3.24 : Matrice de confusion - Augmentation de données et réglage fin - AgrilPlant 61
Figure 3.25 : Interface 1 de l'application Nabet-ID 65
Figure 3.26 : Interface 2 de l'application Nabet-ID 66

Liste des tableaux

Tableau 1-1 : Comparaison entre les applications [4] 15
Tableau 3-1 : Résultat d'apprentissage sur la base de données Pl@ntnet 49
Tableau 3-2 : Résultats d'apprentissage par transfert 50
Tableau 3-3 : Résultats réglage fin 53
Tableau 3-4 : Résultats d'augmentation de données 56
Tableau 3-5 : Résultats de réglage fin et augmentation de données 59

Introduction Générale

Nous vivons sur une planète qui regorge d'espèces végétales. Les chercheurs estiment que notre Terre compte environ 298000 différentes espèces végétales, dont 215644 cataloguées. S'y ajoutent 611000 espèces de champignons et moisissures, dont 43271 faisant l'objet d'une classification. Quant aux espèces de plantes à fleurs (angiospermes), elles varient entre 220000 et 420000 (Tout ça avec seulement 14,1% du total d'espèces découvertes jusqu'à présent) Ce qui rend l'enseignement et l'apprentissage du dictionnaire de la botanique et le vocabulaire des taxons juste Impossible ! Les botanistes ont donc besoin d'un outil qui leur apporterait une aide précieuse pour la classification des espèces. [1] [2] [3]

Par ailleurs, le réchauffement climatique résultant des modifications des écosystèmes et une population en constante croissance ; elle devrait atteindre 9.8 milliards en 2050 [1]. Avec ce dernier facteur, les besoins en nourriture sont de plus en plus importants, d'autant plus que le nombre de personnes végétariennes ne cesse d'augmenter lui aussi. L'identification des plantes est alors cruciale pour le suivi écologique et pour nourrir les sociétés.

En outre, la curiosité des amateurs de jardinage, de plantes, et de la nature, les pousse à comprendre le monde qui les entoure pour prendre soin de leur environnement. Ils ont besoin d'identifier les espèces à l'aide d'outils sophistiqués.

Pour tous ces facteurs et d'autres encore, une application de reconnaissances de plantes devient un outil très précieux. Plusieurs applications ont déjà vu le jour à travers le monde entier, citons : Pl@ntnet, LeafSnap, Flora Incognita … et bien plus encore. Plusieurs chercheurs essaient d'exploiter différentes techniques pour que ces applications puissent reconnaître le plus d'espèces et avec un taux d'exactitude plus élevé. Cela fait de l'identification de plantes par applications un domaine très actif.

L'Algérie ne possède qu'une seule plateforme dans le domaine d'identification de plantes : AdvenAlg[1], c'est une plateforme web dédiée à la reconnaissance des mauvaises herbes.

Pour développer l'application de reconnaissance de plante Nabet-ID, nous nous sommes appuyés sur les réseaux de neurones convolutifs. Notre travail s'articule ainsi sur trois chapitres :

[1] http://www.wikwio.org/alg/index.php

Dans le premier chapitre, nous présentons des applications d'identification de plantes qui existent à travers le monde, ainsi que des études et des publications qui ont été faites sur ce sujet.

Dans le deuxième chapitre, nous exposons quelques architectures de réseaux de neurones convolutifs après en avoir présenté les composantes principales notamment avec la phase d'entraînement.

Le chapitre trois est consacré à la présentation de quelques concepts généraux pour l'implémentation sur mobile.

Le chapitre quatre présente l'implémentation de l'application Nabet-ID sur mobile.

Tout naturellement, nous clôturons ce mémoire par une conclusion générale et des perspectives.

Chapitre 1 : Application de reconnaissance de plantes

1.1 Introduction

Ce premier chapitre est réservé pour la présentation des applications d'identification de plantes existantes, ainsi que quelques études et des publications scientifiques qui ont été faites dans ce domaine.

1.2 Présentation des applications les plus connues

De nombreuses applications existent à travers le monde permettant aux scientifiques et aux amateurs de la nature, qui souhaitent en apprendre un peu plus sur les plantes qui les entourent, d'identifier les feuilles et les plantes à partir d'une photo prise de la plante en question.

1.2.1 LeafSnap

L'application LeafSnap[2] est créée en 2009 par collaboration entre l'Université de Columbia, l'Université du Maryland et les botanistes de la Smithsonian Institution. LeafSnap permet de reconnaître 32000 espèces de plantes à partir d'une photo de la feuille, LeafSnap affiche aussi des informations sur les fleurs, les fruits, les graines et l'écorce de l'arbre, pour donner à l'utilisateur une meilleure compréhension de l'espèce en question. Cependant elle nécessite une connexion internet. De plus, des commentateurs de cette application ont souligné que pour améliorer la précision de l'identification, les photos doivent être prise sur fond blanc, ce qui pourrait être un problème lors d'une balade dans la nature. L'application LeafSnap apprend et ajoute en permanence des informations sur de nouvelles espèces de plantes pour améliorer encore plus ses performances.

[2] https://play.google.com/store/apps/details?id=plant.identification.snap&hl=fr&gl=US

Figure 1.1 : L'application LeafSnap

1.2.2 Pl@ntnet

Pl@ntnet[3] une application française largement connue dans le monde d'identification de plantes, elle a été développée en collaboration avec quatre organismes de recherche français (Cirad, Institut national de la recherche agronomique [INRA], Institut français de recherche en informatique et automatique [Inria] et Institut National de Recherche pour le Développement Durable (IRD)) et le réseau Tela Botanica. Pl@ntnet est accessible en application mobile (par smartphone, tablette Android, ...) ou sous un service accessible depuis un ordinateur via un simple navigateur internet. L'application Pl@ntnet a réussi à identifier 37499 espèces de toutes sortes de plantes vivantes dans la nature : des plantes à fleurs, des arbres, des herbes, des conifères, des fougères, des lianes, des salades sauvages ou bien encore des cactus, grâce aux technologies de l'intelligence artificielle : le deep learning, en plus particulier. Pour améliorer les performances, Pl@ntnet propose à ses utilisateurs du monde entier de contribuer à l'augmentation de sa base de données notamment avec les photos de plantes.

[3] https://play.google.com/store/apps/details?id=org.plantnet&hl=fr&gl=US

Figure 1.2 : L'application Pl@ntnet

1.2.3 Flora Incognita

L'application Flora Incognita[4], lancée en avril 2018, a été développée par l'Université Technique d'Ilmenau et l'Institut Max Planck de Biogéochimie. C'est une application largement utilisée pour l'identification des plantes en Europe. Flora Incognita permet avec les réseaux de neurones convolutifs une identification automatique de 4851 espèces de plantes vasculaires. Elle propose, en outre, une fiche technique qui affiche des informations complémentaires telles que les caractéristiques, la répartition ou le statut de protection.

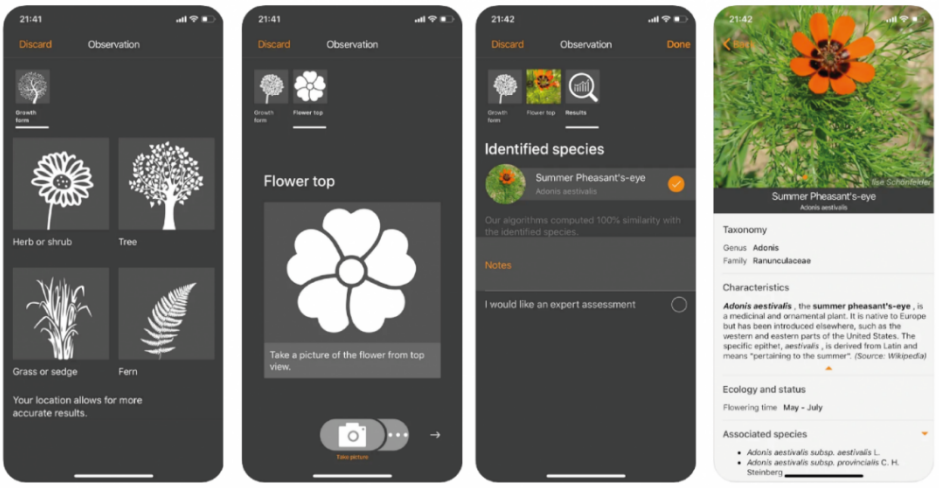


Figure 1.3 : Flora Incognita

[4] https://play.google.com/store/apps/details?id=com.floraincognita.app.floraincognita&hl=fr&gl=US

1.2.4 PlantSnap

L'application PlantSnap[5] permet à ses utilisateurs d'identifier plus de 600000 espèces végétales du monde entier : les espèces de fleurs, feuilles, arbres, champignons, plantes succulentes et cactus. De plus, PlantSnap propose des astuces et des conseils de jardinage, de créer des collections de plantes et de rassembler toutes les découvertes en un seul endroit.

Figure 1.4 : L'application PlantSnap

1.2.5 Comparaison entre les applications

L'application LeafSnap ne peut identifier la plante que par sa feuille. Contrairement aux autres applications.

Les applications Pl@ntnet et Flora Incognita combinent plusieurs images de différents organes végétaux, ce qui leur permet d'améliorer les performances d'identification. [4]

Le tableau ci-dessous [4] présent les scores moyens obtenus pour la première identification par chacune des trois applications pour chaque sous-groupe d'échantillons, que ce soit par type de plante (monocot, herbacée ou ligneuse) ou par partie de plante (fleur, fruit, plante ou feuille), ainsi que le nombre d'échantillons dans chaque sous-ensemble.

[5] https://www.plantsnap.com

Application	Pl@ntnet	Flora Incognita	PlantSnap
Fleur	58	67	42
Fruit	73	33	47
Feuille	42	48	31
Plant	50	56	43
Herbe	49	52	38
Monocotylédone	50	61	31
Boisé	65	61	53
Moyenne	52.1	60.3	39.7

Tableau 1-1 : Comparaison entre les applications [4]

1.3 Pourquoi les applications de reconnaissance de plantes utilisent-elles le deep learning

Si les méthodes traditionnelles d'apprentissage automatique nécessitent d'extraire les features avant d'appliquer l'algorithme de classification, le deep learning ne nécessite aucun prétraitement pour l'extraction de caractéristiques, ce qui permet d'économiser du temps et des efforts. De plus, la puissance du deep learning a impressionné le secteur agricole depuis son utilisation pour la reconnaissance de plantes en 2015, où la précision de l'identification est passée de 45% à 65% tandis que les espèces correctement identifiées ont doublé de 500 à 1000 espèces [2]. Les applications font désormais confiance au deep learning et cette tendance devrait se confirmer pour les années avenir :

Alexis Joly, co-responsable scientifique du projet Pl@ntnet affirme [1] : "Étant du monde de l'informatique et des data sciences, on a identifié très tôt le potentiel du deep learning. Avant 2015, on utilisait d'autres méthodes et puis, soudain, on a vu les progrès spectaculaires qui étaient opérés en analyse d'image. En tant que chercheurs, nous participons et coorganisons

régulièrement un challenge, le Life CLEF, qui permet de comparer les performances de systèmes et… nos systèmes y arrivent en bonne position !"

Il continue : "...nous allons continuer à être de gros utilisateurs de Deep Learning car nous n'avons plus de doute sur l'intérêt de ces technologies."

Colin Chaballier, Carbon Bee AgTech [1] : « Le Deep Learning nous permet d'appréhender la très grande variabilité de conditions et de situations auxquelles est soumise la détection de plantes ou de symptômes. Par exemple, pour la reconnaissance de plantes, il nous faut gérer des cas de carences, des stades phénologiques différents, des différences de taille entre les individus, petits ou grands, biscornus ou pas, avec ombre ou pas, une multitude de variétés, etc… On cherche donc une vraie souplesse et flexibilité d'usage. Ensuite, il répond au type de problématique que nous souhaitons pouvoir traiter, à savoir proposer des algorithmes « tout terrain ». On veut pouvoir aller chercher différents symptômes en embarquant les mêmes capteurs sur un quad et en faisant tourner les mêmes algorithmes. La seule chose qui doit changer, ce sont les données d'entrée, les bases de connaissance mises en œuvre. »

1.4 Etudes et publications scientifiques

Plusieurs études ont été réalisées dans le domaine de l'identification des plantes. Les premières se sont concentrées sur l'utilisation des algorithmes de machine learning comme : les machines à vecteur de support SVM, les k plus proches voisins KNN, les forêts aléatoires...etc. Mais avec la révolution du deep learning, les récentes études se sont tournées vers les CNN pour accomplir au mieux l'identification des plantes.

Wu et al. [5] Présentent une approche modifiée de réseau de neurones pour la reconnaissance des feuilles des plantes. Ils utilisent une chaîne de traitement comprenant : Capturer l'image numérique de la feuille, traiter l'image, extraire les caractéristiques PCA, entraîner le PNN, tester le PNN, afficher et comparer les résultats. La précision obtenue est supérieure à 90% sur 32 types de plantes.

Nghia et al [6] aborde plusieurs problèmes critiques de la classification de riz et des plantes médicinales. Ils proposent une application mobile (Figure 1.5) basée sur une approche d'apprentissage par transfert avec des réseaux neuronaux convolutifs profond. Leurs expérimentations montrent que la combinaison proposée fonctionne bien et atteint la précision de classification de 98,7% avec l'architecture Inception sur un dataset de décoloration de

grain, et 98,5% avec l'architecture MobileNet 1.0-244 sur l'ensemble de données de plantes médicinales.

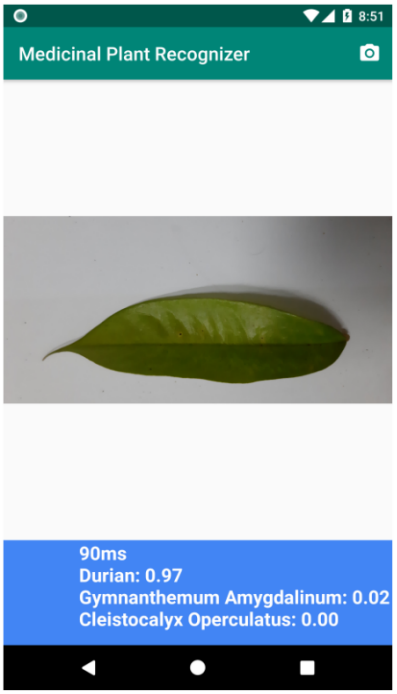


Figure 1.5 : L'application medicinal plant recognizer [6]

T.J. Jassmann et al, [7] dans son article décrit la classification de plante comme problème compliqué même pour les biologistes qualifiés. Il a appliqué les réseaux de neurones convolutifs avec différentes profondeurs pour créer une application mobile (Figure 1.6) capable de distinguer 15 espèces de feuilles avec leur arrière-plan naturel prises par l'appareil. La précision des trois premiers (qu'ils ont obtenu) est égale à 81,6%.

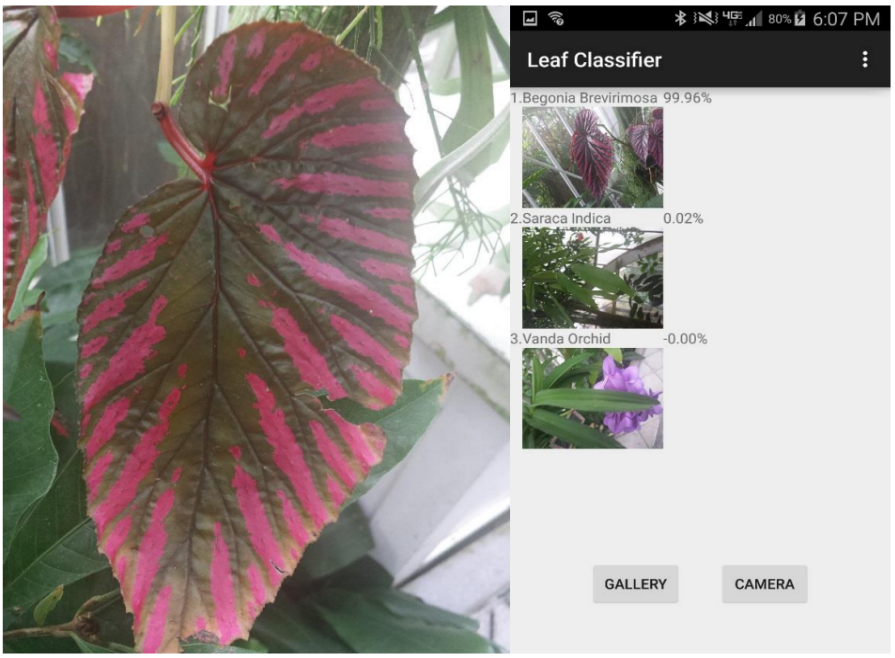


Figure 1.6 : Leaf Classifier [7]

1.5 Conclusion

Dans le présent chapitre, nous avons présenté quelques applications qui existent à travers le monde. Ces applications ont recours au deep learning et aux réseaux de neurones convolutifs pour identifier les plantes. Nous allons voir les réseaux de neurones convolutifs plus en détail dans le chapitre qui suit.

Chapitre 2 : Les réseaux de neurones convolutifs

2.1 Introduction

Ce chapitre expose dans un premier temps les réseaux de neurones convolutifs et leurs composantes principales (les couches de convolution, les couches d'activation, les couches de pooling et les couches entièrement connectées), ainsi que l'entraînement des réseaux de neurones. Dans un second temps, quelques architectures classiques et modernes sont présentées.

2.2 Les réseaux de neurones convolutifs

Les réseaux de neurones convolutifs (Convolutional Neural Network, ConvNets ou CNN) sont un type particulier de réseaux de neurones artificiels. Les CNN sont inspirés du cortex visuel des humains ou bien des animaux. Ils sont conçus pour apprendre automatiquement les caractéristiques des images, contrairement à d'autres méthodes d'apprentissage automatique (Figure 2.1). Il a été prouvé que les CNN sont très efficaces dans des domaines tels que la reconnaissance et la classification d'images. [8]

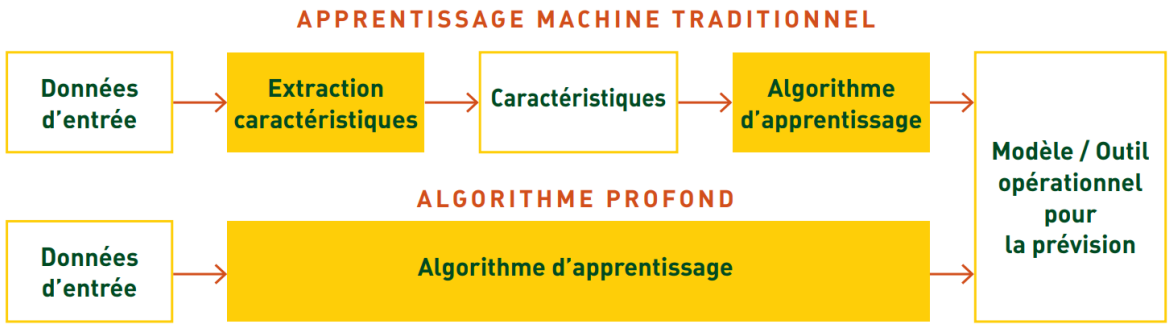


Figure 2.1 : Apprentissage automatique Vs Apprentissage profond [1]

Les architectures CNN se forment des couches suivantes : couches de convolution, couches d'activation, couches de pooling et les couches entièrement connectées. La figure 2.2 illustre une architecture simplifiée des réseaux convolutifs.

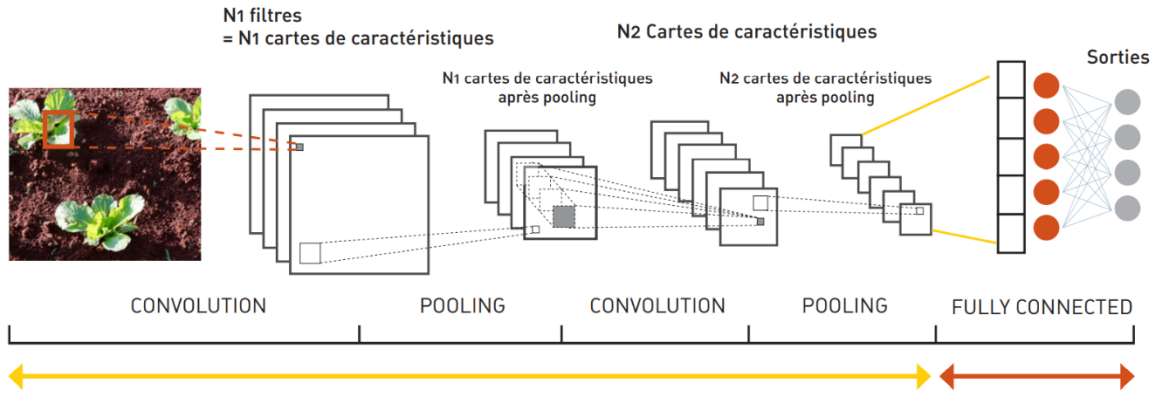


Figure 2.2 : réseau de neurones convolutif [1]

2.2.1 Les couches de convolution

La fonction des couches de convolution est l'extraction des caractéristiques ou features pertinentes des images. Ces couches prennent en entrée une image et font opérer une convolution entre un champ réceptif de l'image d'entrée et un filtre (noyau ou kernel) qui est une petite matrice de poids. Cette opération est appliquée pour toute la matrice image. La figure 2.3 illustre cette opération :

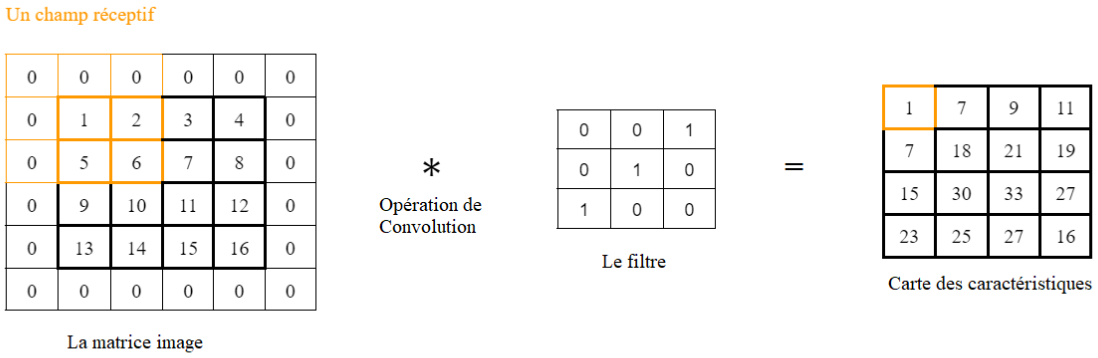


Figure 2.3 : Opération de convolution

Paramètres et hyper-paramètres des couches de convolution

Les paramètres sont des variables apprenables que le réseau de neurones doit optimiser lors de l'entraînement du réseau. Les filtres sont les paramètres des couches convolutives.

Quant aux hyperparamètres, ils représentent des variables qui doivent être définies au préalable, avant l'entraînement du réseau. La « profondeur », la « marge » et le « pas » composent les hyper-paramètres des couches convolutives.

1. La profondeur (d) : Elle représente le nombre de filtres de convolution.

2. La marge ou Padding (p) : C'est une technique qui consiste à ajouter une frontière de la matrice d'image d'entrée. Les valeurs de cette frontière sont le plus souvent des zéros. Le padding est utilisé pour empêcher le filtre de dépasser le cadre de cette matrice. Il existe deux types de padding :

a. Le padding valide : qui correspond à la situation où on ne rajoute pas de frontières. La sortie de padding valide est de taille :

$$(n - f + 1)\times(n - f + 1) \qquad (2.1)$$

b. Same padding : Correspond à la situation où la taille de sortie est égale à la taille de l'entrée :

$$(n + 2 \times p - f + 1) = n \qquad (2.2)$$

$$donc : p = \frac{f-1}{2} \qquad (2.3)$$

3. Le pas ou straid (s) : C'est le nombre de pixels par lesquels la fenêtre se déplace après avoir effectué l'opération de convolution. Taille de sortie est donc :

$$\lfloor n + 2\times p - f + 1 \rfloor \times \lfloor n + 2\times p - f + 1 \rfloor \times d \qquad (2.4)$$

Avec : n : la taille de la matrice image,

f : la taille du filtre,

p : la marge,

s : le pas

Le coût de calcul de la couche de convolution :

$$Coût = Tf \times Ts \qquad (2.5)$$

Avec $Tf = f \times f \times nombre\ de\ canaux$ et Ts est la taille de sortie.

Exemple : Supposons que l'image d'entrée est en RGB et de taille 6x6. Faisons une 5 fois convolution avec un filtre de 3x3x3. Avec un padding de type valide et un pas de 1.

- La sortie de la couche de convolution sera donc :

$$\lfloor \frac{6+2\times0-3}{1} + 1 \rfloor \times \lfloor \frac{6+2\times0-3}{1} + 1 \rfloor \times 5 = 4\times4\times5 \qquad (2.6)$$

- Le coût de calcul de cette exemple égale a :

$$(3 \times 3 \times 3) \times (4 \times 4) \times 5 = 2160 \qquad (2.7)$$

2.2.2 Les couches d'activation (activation layers)

Les couches d'activation (également appelées : couches de correction) sont appliquées par les CNN à chaque sortie de couche de convolution. Les couches d'activation sont employées pour permettre au réseau de décider si un signal doit être passé à la couche suivante ou pas pour éviter le problème de disparition ou de saturation du gradient (vanishing or saturation gradient): La disparition du gradient se produit lorsque le réseau de neurones devient de plus en plus profond, le gradient devient de plus en plus petit, ce qui empêche le réseau de se former, car les poids du réseau de neurones ne peuvent pas être modifiés correctement.

Pour la réalisation de cette tâche les couches d'activations s'appuient sur des fonctions d'activation. Mais il faut bien choisir la fonction d'activation, sinon on se retrouvera avec un problème d'explosion ou de disparition de gradient.

La fonction tangente hyperbolique

La formule de la fonction d'activation tangente hyperbolique (hyperbolic tangent, tanh) est :

$$f(z) = \frac{1 - e^{-2z}}{1 + e^{-2z}} \qquad (2.8)$$

La sortie de la fonction tangente hyperbolique (Figure 2.4) est entre -1 et 1, centrée sur 0. L'optimisation est plus facile avec cette fonction. Cependant elle peut causer l'explosion de gradient ; Les gradients deviennent trop grands faisant diverger l'algorithme par des poids très grands.

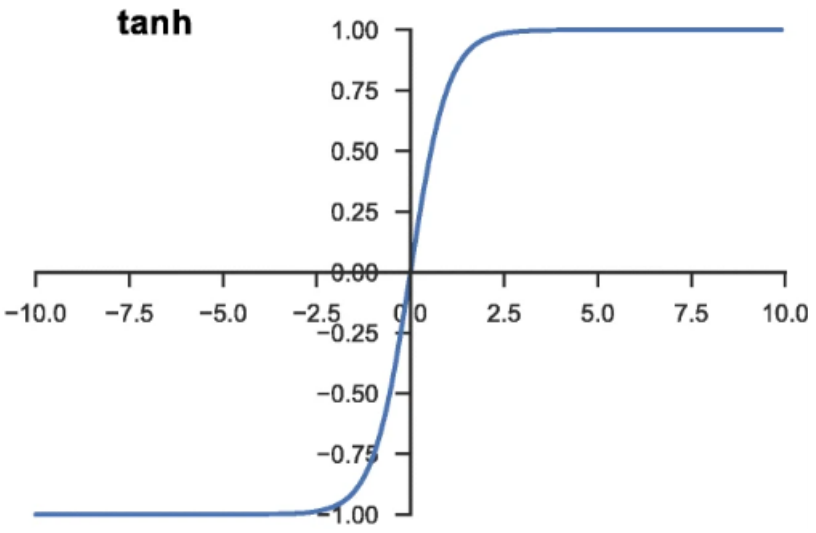


Figure 2.4 : La fonction tanh [9]

La fonction d'activation ReLU (Rectified Linear Units)

ReLU (Rectified Linear Units) est la fonction d'activation la plus simple : elle ne laisse passer que les valeurs positives ou nulles. De plus, elle est la fonction d'activation la plus utilisée, car elle améliore la convergence de six fois. Un autre avantage de cette fonction est qu'elle évite et corrige le problème d'explosion de gradient. Sa formule mathématique est :

$$ReLU(z) = max(0, z) \qquad (2.9)$$

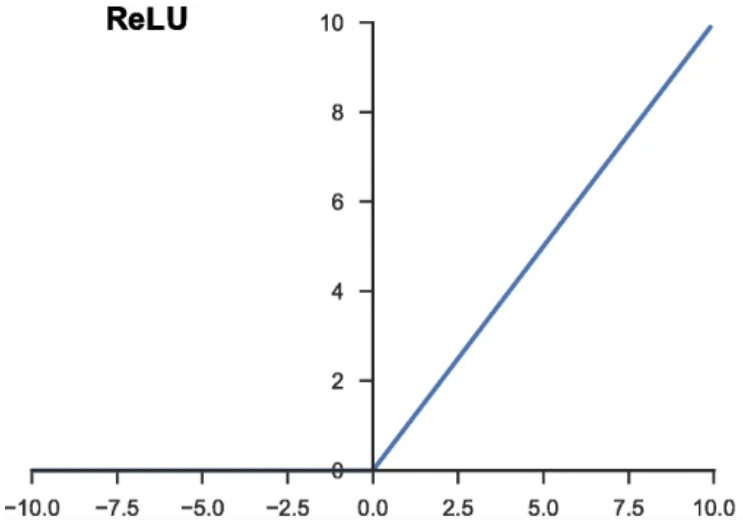


Figure 2.5 : La fonction ReLU [9]

Plusieurs autres fonctions d'activation existent dans la littérature, mais la fonction ReLU reste la fonction d'activation la plus utilisée, car elle donne dans une majorité de cas un meilleur résultat.

2.2.3 Les couches de Pooling (mise en commun)

La couche Pooling vise à compresser l'image tout en préservant ses caractéristiques. Avec cette compression, la complexité spatiale sera optimisée et il y aura moins de paramètres et de calcul dans le réseau. Pour les images en couleur RGB (red, green and blue), chaque canal est traité indépendamment. Les types de pooling les plus utilisés sont :

Max-pooling : qui consiste à renvoyer la valeur maximale du champ réceptif.

Avg-pooling : elle renvoie la moyenne des valeurs du champ réceptif.

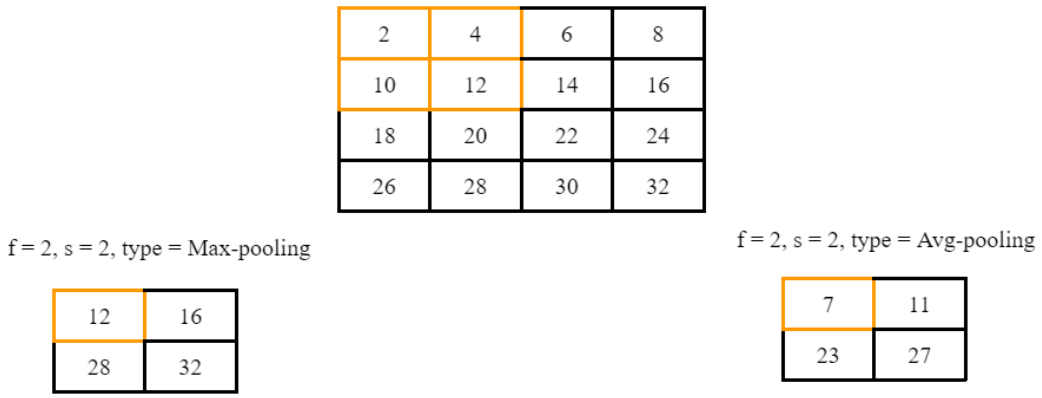


Figure 2.6 : Opération de pooling

Les paramètres et hyper-paramètres des couches de pooling

La couche de pooling n'a aucun paramètre à apprendre.

Les hyper-paramètres des couches de pooling sont le type de pooling, la taille de la fenêtre f, le pas s, et la marge p.

La taille de la sortie des couches de pooling :

$$\lfloor \frac{n+2\times p-f}{s} + 1 \rfloor \times \lfloor \frac{n+2\times p-f}{s} + 1 \rfloor \qquad (2.10)$$

Avec : n : la taille de la matrice image,

f : la taille de la fenêtre,

p : la frontière rarement utiliser dans les couches de pooling,

s : le pas

Exemple : Supposons que l'image d'entrée de la couche de pooling est 4x4. Avec une fenêtre de taille 2x2 et un pas de 2 de on obtient une image

$$\lfloor \frac{4+2\times 0-2}{2} + 1 \rfloor \times \lfloor \frac{4+2\times 0-2}{2} + 1 \rfloor = 2\times 2 \qquad (2.11)$$

2.2.4 Les couches entièrement connectées

La couche entièrement connectée (fully connected layer, FC) représente la toute dernière couche dans un CNN. Son but est d'aplatir le résultat de la couche précédente en un seul vecteur afin de classifier l'image passée en entrée du réseau, en lui affectant des valeurs probabilité. Contrairement aux autres couches, les neurones dans la couche entièrement connectée ont des connexions vers toutes les sorties de la couche précédente. Pour la tâche de classification, la couche entièrement connectée fait appel à une fonction d'activation, qui doit être choisie selon le type de problème dont il est question. La fonction d'activation aide à générer une probabilité telle que des scores pour chaque classe.

La fonction d'activation Sigmoïde est un exemple de fonction d'activation utilisée par la couche entièrement connectée quand il s'agit de problèmes de classification binaire. Sa formule est :

$$Sigmoïde(z) = \frac{1}{1+e^{-z}} \qquad (2.12)$$

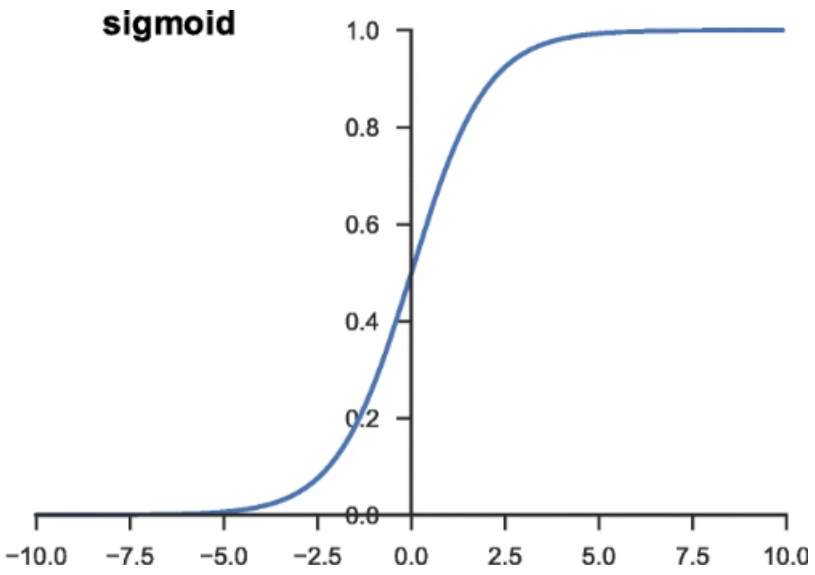


Figure 2.7 : La fonction Sigmoïde [9]

La fonction Softmax est un exemple de fonction d'activation, utilisée par la couche entièrement connectée, pour les problèmes de classification multiples (Figure 2.8). Sa formule :

$$Softmax(z) = \frac{e^{z}}{\sum_{j} e^{z_j}} \qquad (2.13)$$

Où Z est l'ensemble des valeurs des neurones dans la couche de sortie avant l'application de la fonction d'activation.

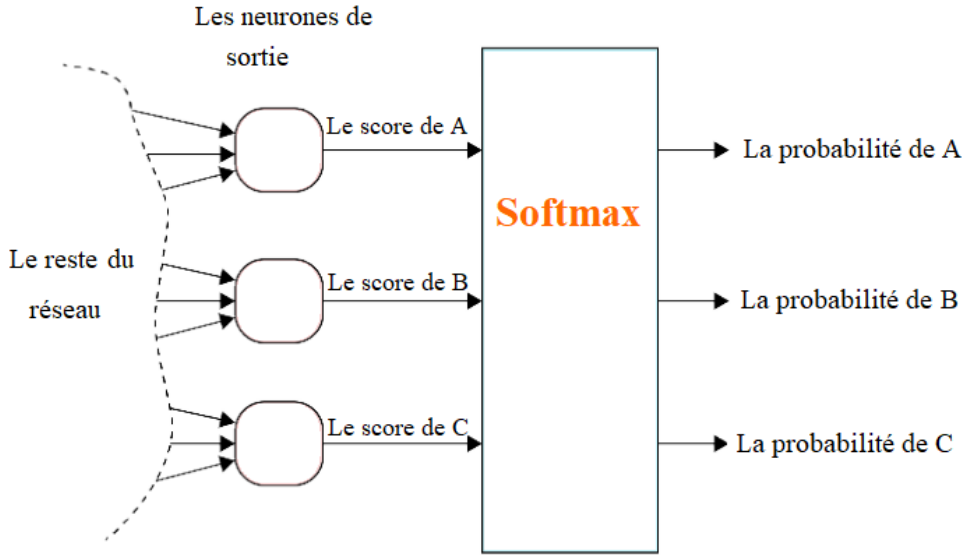


Figure 2.8 : Softmax [9]

A, B et C représentent les classes de la base de données.

Les paramètres et hyperparamètres de couches entièrement connectées

Les paramètres de la couche entièrement connectée sont les poids (Weights).

Les hyperparamètres de couche entièrement connectée sont le nombre de poids.

2.3 Entraînement d'un réseau

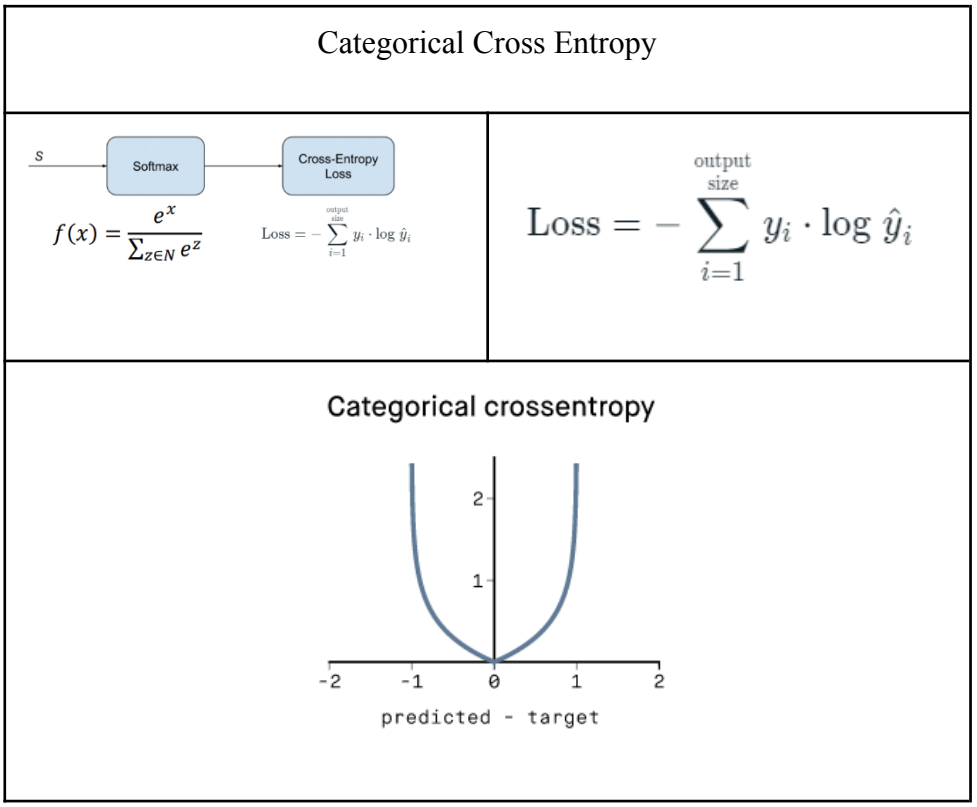
L'entraînement d'un réseau consiste à déterminer les paramètres apprenables : les filtres dans les couches de convolutions et les poids dans les couches entièrement connectées. Ces paramètres sont recherchés comme solution d'un problème d'optimisation, à savoir minimiser la fonction perte (loss function). Cette fonction mesure la différence entre les prédictions de sorties et les labels donnés.

2.3.1 La fonction de perte

Une fonction de perte est utilisée pour optimiser les valeurs des paramètres dans un modèle de réseau neuronal. Le type de fonction utilisée doit être déterminé en fonction des tâches données. Par exemple, la fonction erreur quadratique pour les problèmes de régression et la fonction de perte couramment utilisée pour la classification multiclasse est l'entropie croisée (cross entropy).

La fonction entropie croisée catégorielle (Categorical Cross-Entropy loss)

L'entropie croisée catégorielle est une fonction de perte utilisée dans les problèmes de classifications multi-classes et avec des problèmes qui n'acceptent qu'une seule classe comme sortie. Tout comme la fonction d'activation Softmax, c'est d'ailleurs la raison pour laquelle elle est aussi appelée : La fonction de perte Softmax (Softmax Loss). [10]



2.3.2 Apprentissage par descente de gradient

La descente de gradient est couramment utilisée comme algorithme d'optimisation de la fonction de perte. Le gradient de la fonction de perte fournit la direction dans laquelle la fonction a le taux d'augmentation le plus élevé. Le gradient est, mathématiquement, une dérivée partielle de la perte par rapport à chaque paramètre apprenable. Chaque paramètre apprenable est mis à jour dans la direction négative du gradient avec une taille de pas arbitraire déterminée sur la base d'un hyperparamètre appelé taux d'apprentissage (learning rate).

La forme la plus simple de l'algorithme de descente de gradient est illustré ci-dessous :

1.Initialiser avec x0 (au hasard)

2. Répéter (Jusqu'à convergence) :

$$w = w - \alpha \times \frac{\partial L}{\partial w}$$

Avec : w : les paramètres apprenables,

α le taux d'apprentissage,

Le gradient $\frac{\partial L}{\partial w}$

a. Choix du taux d'apprentissage (α)

Le taux d'apprentissage α, détermine la vitesse de convergence du processus. Trop petit, l'apprentissage est lent et le risque de tomber dans un minimum local est important, trop grand, l'algorithme risque de diverger. (Figure 2.9)

Les valeurs de α peuvent être adaptées au fil des itérations : "grandes" au début pour accélérer la convergence, "faibles" à la fin pour améliorer la précision.

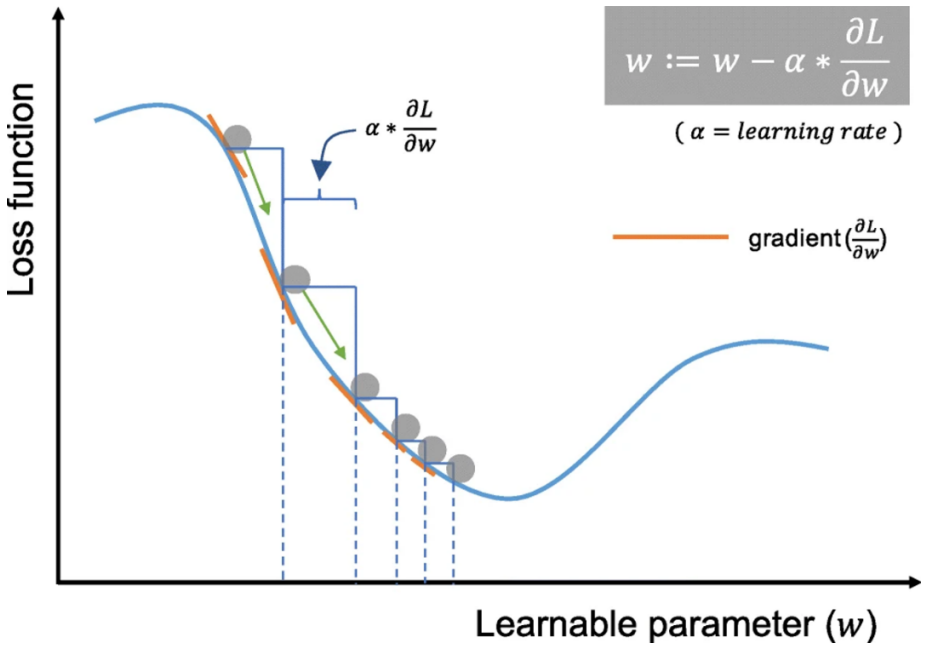


Figure 2.9 : La fonction de perte en fonction du taux d'apprentissage [9]

b. La méthode de descente de gradient stochastique

Le calcul dans l'algorithme de descente du gradient peut devenir très coûteux en temps et en capacité mémoire. Une solution à ce problème est la méthode de descente stochastique (stochastic gradient descent, SGD). C'est une variante de la méthode du gradient applicable lorsque la fonction de perte s'écrit comme une somme de fonctions dérivables. Le processus de descente est alors réalisé sur des lots de données tirés aléatoirement du jeu de données global à chaque itération.

Algorithme de descente de gradient stochastique :

1.Initialiser avec x0 (au hasard)

2. Répéter (Jusqu'à convergence) :

$$w = w - \sum_{i=1}^{B} \frac{\partial L_i}{\partial w}, \; Best\ la\ taille\ du\ lot$$

c. La méthode Adam

L'optimiseur Adam vient de **ada**ptive **m**oment estimation, Il s'agit d'une extension de la descente de gradient stochastique. Dans cet algorithme d'optimisation, les moyennes courantes des gradients et des seconds moments des gradients sont utilisées pour calculer les taux d'apprentissage adaptatifs pour chaque poids du réseau de neurones. Cet algorithme a l'avantage d'être relativement robuste et permet d'adapter automatiquement le taux d'apprentissage au cours de l'apprentissage pour chaque poids.

2.3.3 Époques, lots et itérations

A l'entraînement du réseau on ne peut pas transmettre l'intégralité de la base de données en une seule fois ; la base de données sera divisée en plusieurs lots (batchs). Le nombre de lots est appelé itération. Une époque (epoch) est terminée lorsque tous les lots sont passés au réseau, càd toute la base de données. Exemple : Si la base de données contient 256 images et que la taille du lot est 64. Il faudra 4 itérations pour terminer 1 époque.

2.3.4 Sous-apprentissage, sur-apprentissage et régularisation

Le sous-apprentissage (Underfitting) est une situation où le modèle de réseau neuronal n'arrive pas à trouver les relations et les corrélations dans les données d'entraînement et leurs labels, ce qui induit une erreur élevée pour l'ensemble d'entraînement (training error).

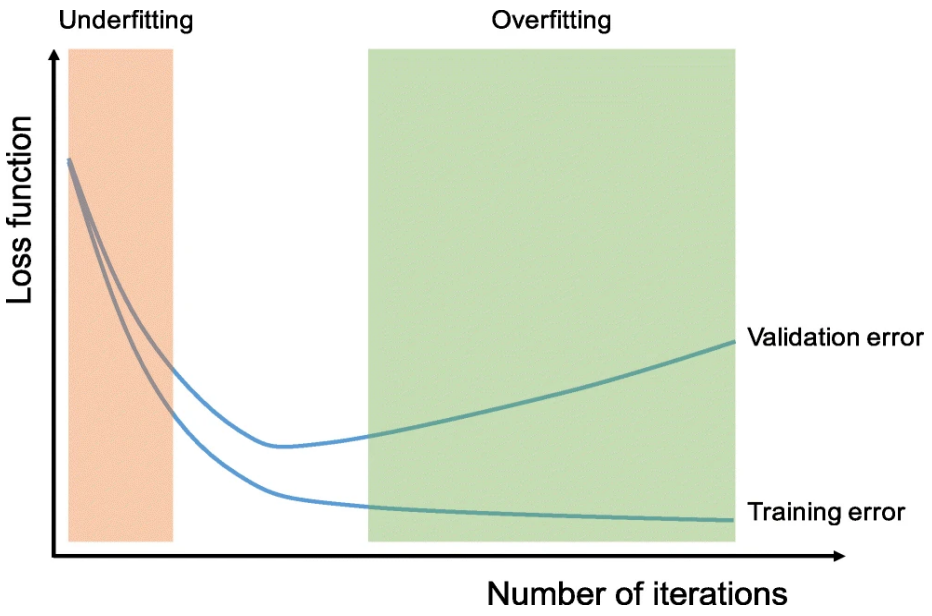


Figure 2.10 : Courbe de validation et d'entraînement pour la fonction de perte [9]

Plusieurs méthodes ont été proposées pour régler le sous-apprentissage du modèle :

1. Obtenir plus de données et augmenter la taille de l'ensemble d'entraînement.
2. Augmenter la complexité architecturale.
3. Augmenter la taille ou le nombre de paramètres dans le modèle.
4. Augmenter le nombre d'époques et le temps d'apprentissage jusqu'à ce que la fonction de perte soit minimisée.

Le sur-apprentissage (overfitting), ou encore surajustement, est l'un des principaux défis des réseaux neuronaux. Il fait référence à une situation où un modèle apprend des régularités statistiques spécifiques à l'ensemble d'apprentissage, par conséquent, il fonctionne moins bien sur un nouvel ensemble de données. Dans ce cas, le modèle n'est pas généralisable à des données jamais vues auparavant. Si le modèle fonctionne bien sur l'ensemble d'apprentissage par rapport à l'ensemble de validation, le modèle a probablement été sur ajusté aux données d'apprentissage. (Figure 2.10)

Plusieurs méthodes de régularisation ont été proposées pour minimiser le sur-apprentissage :

1. La régularisation par élargissement de base de données
   La meilleure solution pour réduire le surajustement est d'obtenir plus de données d'entraînement. Un modèle formé sur un ensemble de données plus large généralise généralement mieux, bien que cela ne soit pas toujours réalisable.
2. La régularisation avec dropout

Dropout est une technique de régularisation pour les modèles de réseaux de neurones proposée par [11]. Dropout est une technique où des neurones sélectionnés au hasard sont ignorés pendant l'entraînement (Figure 2.11). Cela signifie que leur contribution à l'activation des neurones en aval est temporairement supprimée. Les neurones non déconnectés ne peuvent pas compter sur la présence des autres neurones, ils seront donc obligés d'apprendre des fonctionnalités plus robustes. Le dropout double à peu près le nombre d'itérations nécessaires pour converger, mais règle le problème de sur-apprentissage.

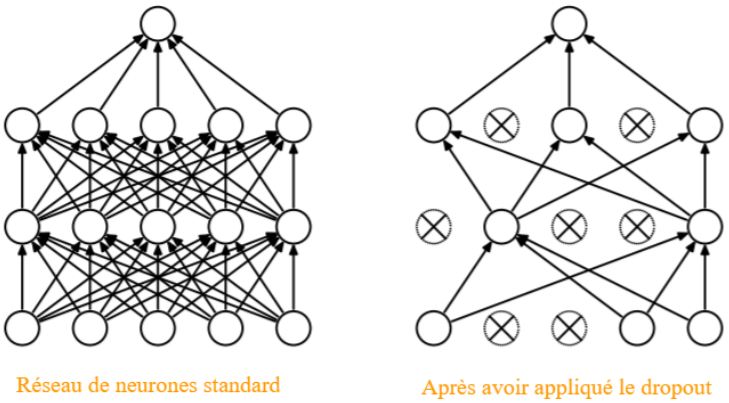


Figure 2.11 : Dropout [11]

3. La normalisation par lots (batch normalization) est une couche supplémentaire qui normalise de manière adaptative les valeurs d'entrée de la couche suivante, atténuant le risque de sur-apprentissage et améliorant le flux de gradient à travers le réseau, permettant des taux d'apprentissage plus élevés et réduisant la dépendance à l'initialisation.
4. L'augmentation des données c'est un processus de modification des données d'entraînement par des transformations aléatoires, telles que le retournement, la traduction, le recadrage, la rotation et l'effacement aléatoire, de sorte que le modèle ne verra pas exactement les mêmes entrées pendant les itérations d'entraînement. Nous détaillons plus l'augmentation de données en chapitre 3.
5. La réduction de la complexité architecturale.

Finalement, tester les performances du modèle final sur un ensemble de tests distinct et idéalement sur des ensembles de données de validation externes est crucial pour vérifier la généralisabilité du modèle.

2.4 Architectures des réseaux de neurones convolutifs

2.4.1 RNC classiques

Les architectures des réseaux classiques se composent d'un empilement de couches de convolutions, de couches de pooling et de couches entièrement connectées, de façon à améliorer la prédiction du réseau.

a.LeNet-5

Conçu par Yann LeCun en 1989, l'architecture LeNet-5 est constituée de sept couches : les deux premières couches convolutives sont suivies d'une couche de pooling, la dernière couche de convolution est suivie de deux couches entièrement connectées qui effectuent la tâche de classification. Le-Net-5 compte 60 mille paramètres. [12]

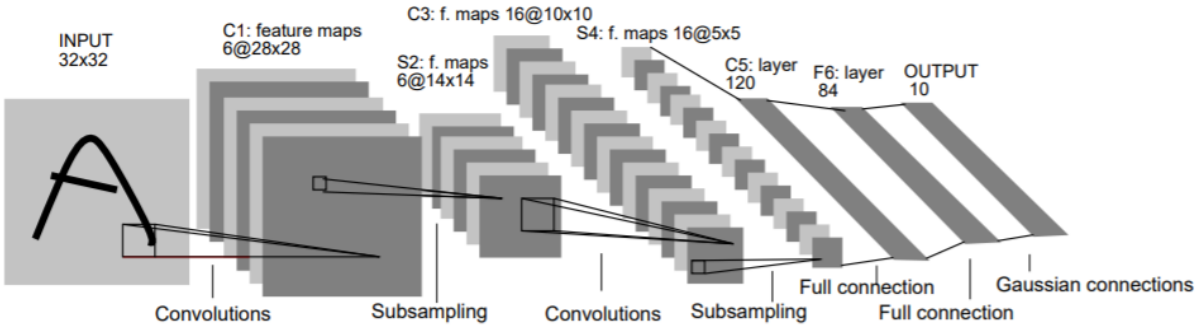


Figure 2.12 : L'architecture leNet-5 [12]

b.AlexNet

L'architecture AlexNet [13] conçu par Alex Krizhevsky a remporté la compétition d'imageNet ILSVRC-2012 (ImageNet Large Scale Visual Recognition Challenge), qui est une compétition annuelle où les équipes de recherche compétitrices ont pour mission de décrocher le taux d'erreur le plus faible sur le jeu de données ImagNet. ImageNet est parmi les plus grandes bases de données au niveau mondial. Elle compte plus de 15 millions d'images de haute résolution correspondant à environ 22000 catégories.

Alex Krizhevsky n'a pas seulement remporté la compétition d'imageNet ILSVRC-201, il a bouleversé le monde de deep learning avec son architecture ; c'est l'architecture révolutionnaire du deep learning. Alexnet est une version plus profonde et beaucoup plus large du LeNet, avec un total de 60 millions de paramètres, elle dispose de huit couches : les cinq premières sont des couches convolutives et les trois dernières sont des couches entièrement connectées, avec au milieu des couches de pooling et d'activation. Les progrès

matériels des processeurs graphiques (GPU) sont derrière le grand succès d'AlexNet. Les contributions principales de l'architecture AlexNet sont :

1. L'utilisation de la fonction d'activation : ReLu.
2. Utilisation de type Max-pooling au lieu de Avreg-pooling.
3. L'utilisation de la technique dropout

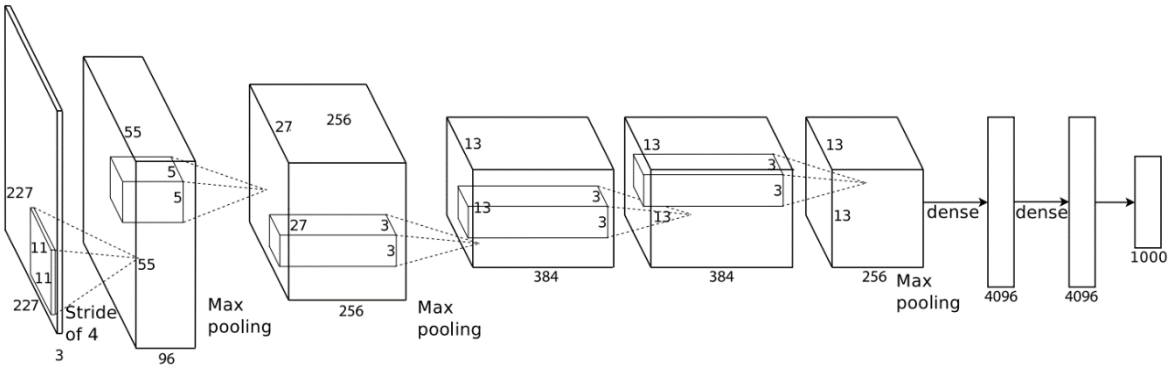


Figure 2.13 : Architecture AlexNet [13]

c.VGG

Après AlexNet l'architecture VGG a remporté la compétition ILSVRC-2014, développée par les chercheurs du Visual Graphics Group à Oxford (d'où son nom VGG). VGG a une architecture pyramidale et assez uniforme avec des filtres de convolutions de tailles 3x3, un pas de 1, VGG utilise le type de pooling : Max-pooling de taille 2x2 avec un pas de 2. Le nombre de filtres est doublé dans cette architecture, alors que la hauteur et la largeur diminuent d'un facteur de 2.

VGG a deux versions : VGG-16 avec 16 niveaux de profondeur, et VGG-19 avec 19 niveaux de profondeur. [14]

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

Figure 2.14 : Architecture VGG [14]

2.4.2 RNC modernes

Contrairement aux architectures de réseaux classiques, les architectures modernes explorent diverses nouvelles techniques pour construire leurs modèles, ces techniques leur permettent une meilleure prédiction et un apprentissage plus efficace.

a.Resnet

Présenté par Shaoqing Ren, Kaiming il, Jian Sun et Xiangyu Zhang, des chercheurs de Microsoft Research, en 2015. La mission de l'architecture Resnet est de créer des réseaux de neurones beaucoup plus profonds car théoriquement, plus le réseau de neurones est profond, meilleure est la prédiction. Mais entre la théorie et la pratique, il existe un obstacle majeur : le problème de la disparition/explosion des gradients (vanishing/exploding gradients). Resnet a résolu ce problème par le principe de bloc résiduels (Residual Blocks). La figure ci-dessous, issue du papier original : Deep Residual Learning for Image Recognition montre les résultats

d'une étude comparative entre un réseau simple et ResNet : La précision du réseau simple à 34 couches commence à saturer plus tôt que la précision de ResNet. [15]

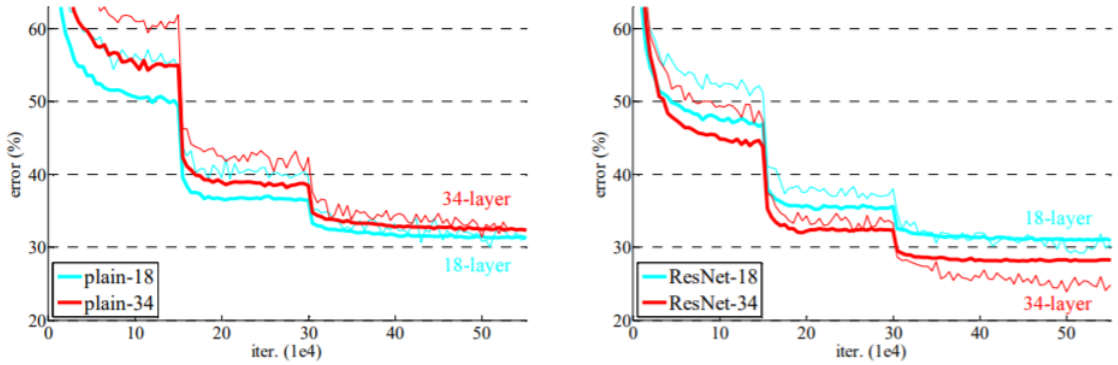


Figure 2.15 : Réseau simple Vs Resnet [15]

Le bloc résiduel, représenté dans la figure ci-dessous, est défini par l'équation : y = F(x) + x ou des connexions résiduelles entre la sortie de la couche précédente sont additionnés à la sortie de la couche courante.

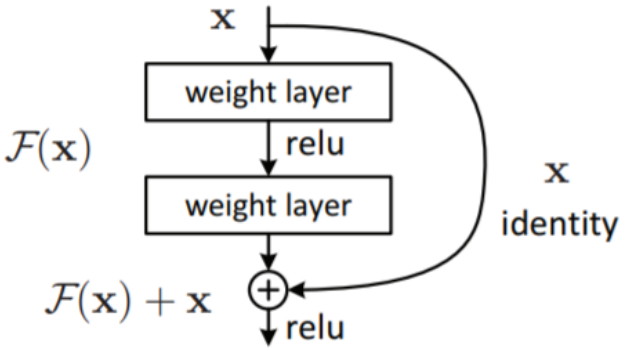


Figure 2.16 : Bloc résiduel [15]

b.Inception

L'architecture Inception [16], aussi connue sous le nom de GoogLeNet, est proposée par des développeurs de Google, en 2014. L'objectif étant de réduire la consommation de ressources des réseaux de neurones. Elle utilise pour cela le principe de convolution 1 par 1 (Networks in Networks), qui lui permet de réduire la profondeur. Ainsi, le coût de calcul sera minimisé également (Christian Szegedy, 2014). La figure 2.17 montre un exemple la réduction de profondeur par le principe de convolution 1 par 1 :

Supposant que :

La taille de la matrice image : n = 28 ;

la taille du filtre de convolution 1 par 1 : f = 1 ;

la frontière : p = 0 ;

le pas : s = 1.

La taille de la sortie est donc :

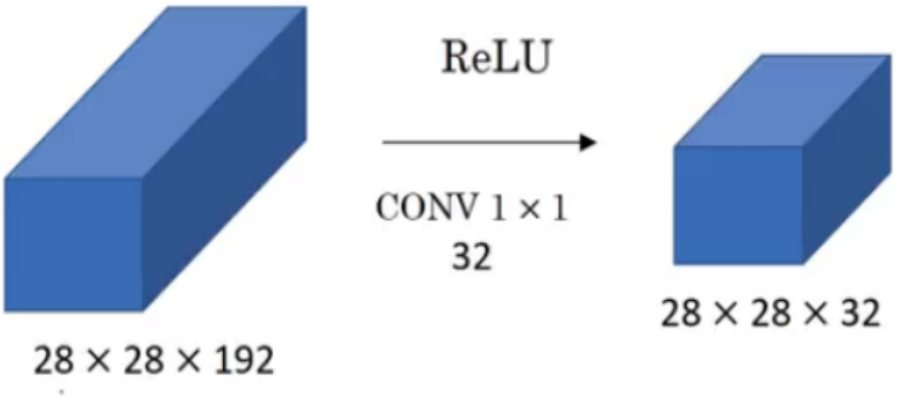


Figure 2.17 : Convolution 1 par 1 [17]

Cette architecture utilise Le Max pooling avec un padding de type : same, cela pour préserver les dimensions de la matrice d'entrée (pour que la sortie puisse être correctement concaténée)

c.MobileNet

Avril 2017, MobileNet [18] est conçue au profit des appareils mobiles et des systèmes embarqués à faible empreinte mémoire. MobileNet vise donc à alléger la taille du modèle, économiser les calculs et diminuer le nombre de paramètres. Pour cela, l'architecture MobileNet divise la couche de convolution en modules de convolutions séparables en profondeur : Convolution spatiale en profondeur (Depthwise Convolution) et convolution ponctuelle (Pointwise Convolution).

1. Convolution spatiale en profondeur est une forme de convolution factorisées qui applique un filtre sur chaque canal d'entrée séparément, contrairement à la convolution classique qui applique un filtre sur l'ensemble des canaux.
2. Convolution ponctuelle applique une convolution 1x1 pour changer la dimension en augmentant la profondeur de la matrice de l'image.

La figure ci-dessous montre comment une convolution standard (a) est factorisée en une convolution en profondeur (b) et une convolution ponctuelle 1 × 1 (c)

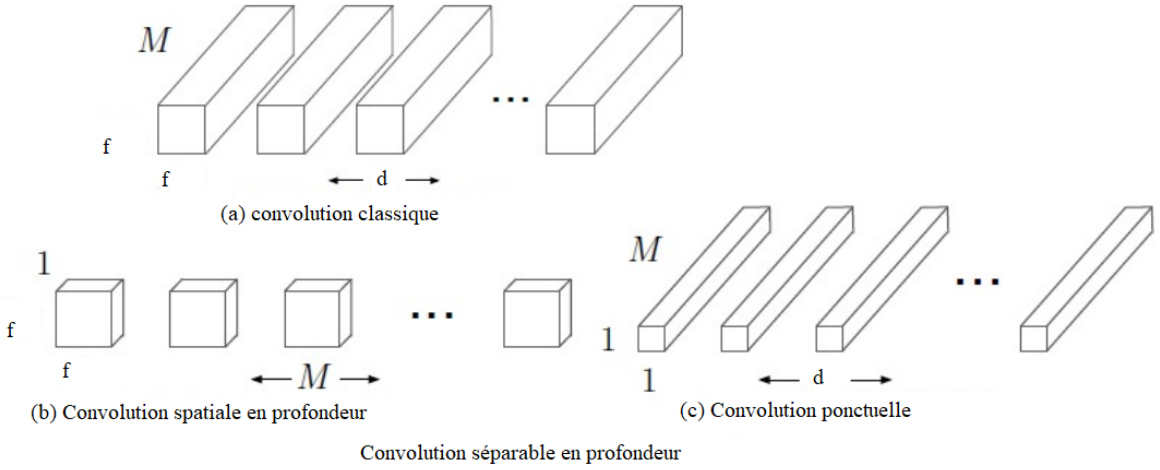


Figure 2.18 : Convolution séparable en fonction de la profondeur [18]

Avec les convolutions séparables en fonction de la profondeur, MobileNet a non seulement réussi à réduire drastiquement la taille de calcul et la taille du modèle, mais aussi le temps de calcul, tout en ayant un faible impact sur la précision.

Si on reprend l'exemple du 2.2.1. La sortie de la convolution spatiale en fonction de la profondeur sera égale à :

$$\lfloor \frac{6+2\times0-3}{1} + 1 \rfloor \times \lfloor \frac{6+2\times0-3}{1} + 1 \rfloor \times 3 = 4\times4\times3 \quad (2.14)$$

Ensuite La sortie de la Convolution ponctuelle égale à :

$$\lfloor \frac{4+2\times0-1}{1} + 1 \rfloor \times \lfloor \frac{4+2\times0-1}{1} + 1 \rfloor \times 5 = 4\times4\times5 \quad (2\text{-}15)$$

Le coût de calcul de convolutions séparables en profondeur est égal au coût de calcul de convolution spatiale en profondeur ajouté au coût de calcul de la convolution ponctuelle.

- Le coût de calcul de la convolution spatiale en profondeur sur cette exemple égale à

$$(3\times3)\times(4\times4)\times3 = 432 \quad (2.16)$$

- Ainsi Le coût de calcul de convolution spatiale en profondeur sur cette exemple égale à

$$(1\times1\times3)\times(4\times4)\times5 = 240 \quad (2.17)$$

- Le coût de calcul de convolutions séparables en profondeur :

$$432 + 240 = 672 \quad (2.18)$$

Pour cette exemple rapport du coût est de 31%

$$\left(\frac{672}{2160} = 0.31\right) \quad (2.19)$$

D'une manière générale, selon le gain de temps avec les convolutions séparables en profondeur par rapport à une convolution classique est :

$$\frac{1}{d} + \frac{1}{f^2} \quad (2.20)$$

d.MobileNetV2

La contribution principale de la version 2 de la série des MobileNet est l'introduction de nouveau module de couche : le résidu inversé (Inverted Residual block) avec goulot d'étranglement linéaire. [19]

- Les résidus inversés comme les blocs de goulots d'étranglement, aident à calculer les activations (ReLU) plus efficacement tout en conservant plus d'informations après l'activation. Sauf que la conception des résidus inversés est considérablement plus économe en mémoire et fonctionne légèrement mieux. Les résidus inversés sont visualisés plus bas.
- Goulot d'étranglement linéaire est utilisé pour conserver l'information.

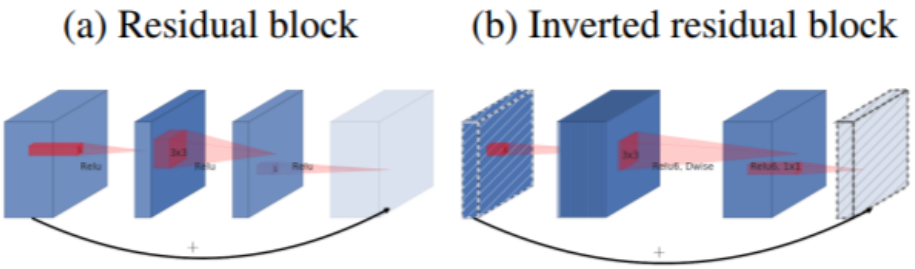


Figure 2.19 : Le résidu inversé [19]

Carlo Lepelaars [20] dans ses expérimentations, sur un dataset de pathologie des plantes, conclut que MobileNet V2 est le meilleur choix parmi tous les modèles de ces expériences si la mémoire de travail est très limitée, ceci justifie le choix de cette architecture dans ce travail.

2.5 Conclusion

Dans ce chapitre nous avons présenté les composants des CNN, à savoir les couches de convolution, couches d'activation, couches de pooling et les couches entièrement connectées.

On a ensuite expliqué les notions importantes relatives à l'entraînement des réseaux de neurones. Enfin, nous avons exposé quelques architectures classiques et modernes des CNN.

Chapitre 3 : Implémentation

4.1 Introduction

Avec ce dernier chapitre nous allons présenter les résultats des expérimentations qu'on a obtenu, ainsi que le déploiement de notre application Nabet-ID sur mobile, et quelques tests.

4.2 Matériel, langage et outils

L'implémentation et la compilation du code source ont été faites avec Anaconda sous Jupyter Notebook.

- Anaconda[6] est une distribution libre, open source, multiplateforme, contenant plusieurs packages nécessaires pour faire du deep learning.
- L'application web Jupyter[7] est utilisée pour faire des programmes avec plus de 40 langages de programmation dont Python. Cette application offre une expérience simple, rationalisée et centrée sur les documents.

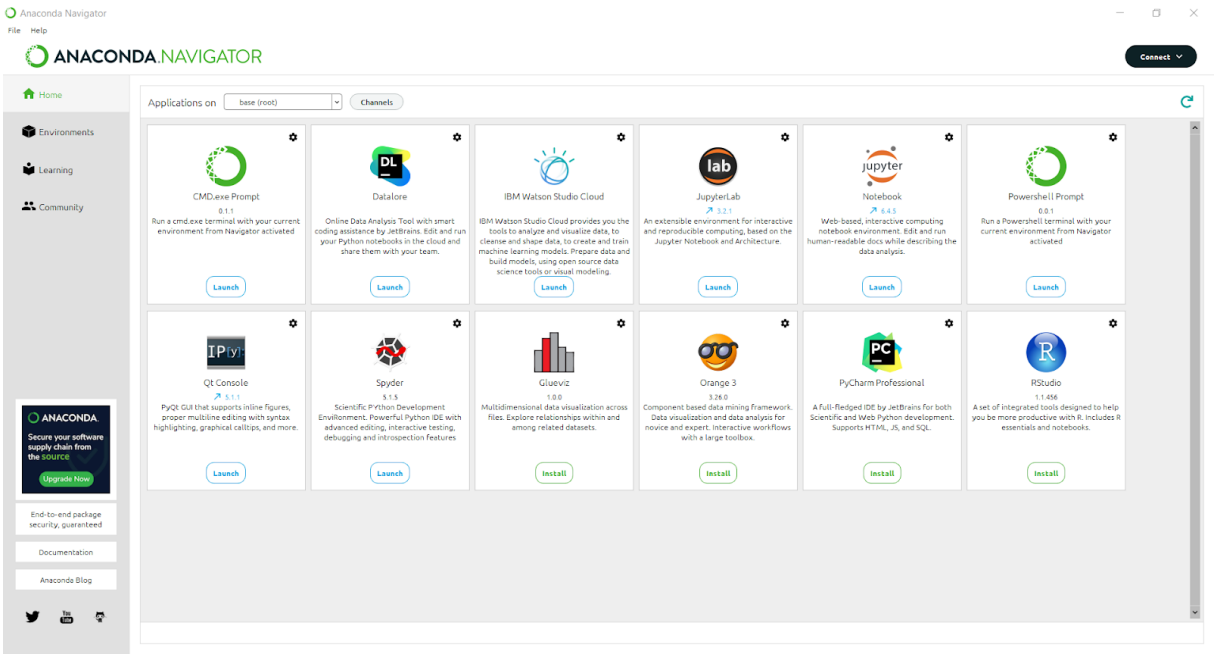


Figure 3.1 : Anaconda navigator

- Sur une machine qui présente les caractéristiques ci-après :
  - Système d'exploitation : Windows 11 ;
  - Type d'architecture du système d'exploitation : 64 bits ;

[6] Anaconda | The World's Most Popular Data Science Platform
[7] https://jupyter.org/

- Processeur : AMD Ryzen 5 3550H with Radeon Vega Mobile Gfx ;
- Mémoire vive : 16 GB ;
- 2 Cartes graphiques :
  - Radeon RX 560X
  - AMD Radeon(TM) Vega 8 Graphics

o Azure Machine Learning est un service cloud proposé par la plateforme Azure destiné aux développeurs. Ce service permet de créer des modèles de machine/deep learning, de les déployer et de les gérer.

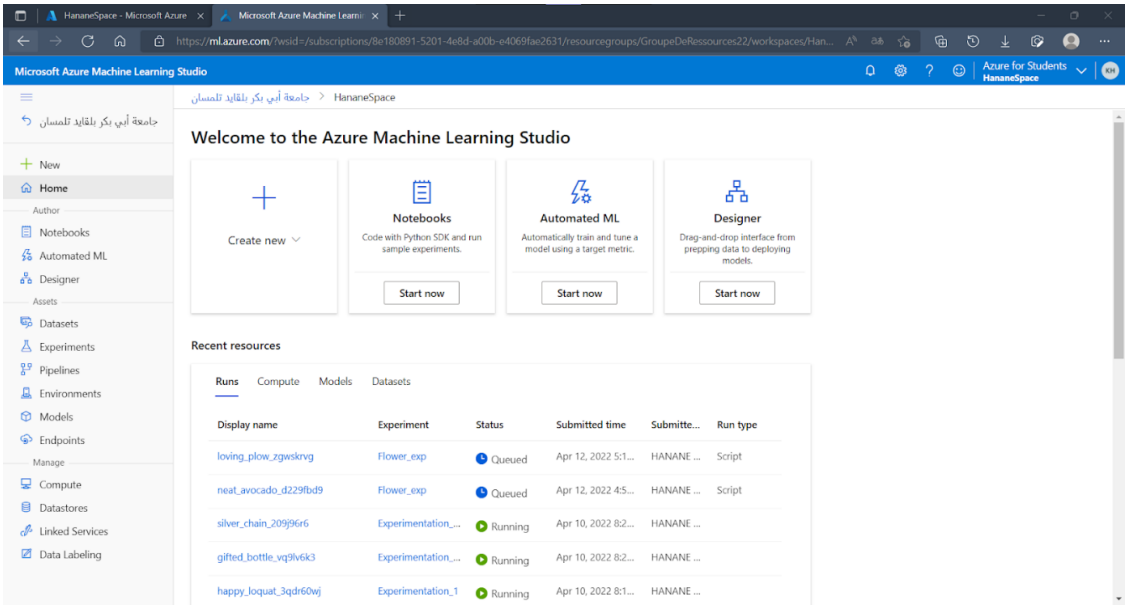


Figure 3.2 : Azure Machine Learning

Les expérimentations ont été effectuées en utilisant :

o Le langage de programmation **Python** qui est un langage de haut niveau interprété ; le script python s'exécute directement sans être compilé avant l'exécution. Python est très sollicité.

o **Tensorflow**[8] : Conçu et développé par Google, Tensorflow est une bibliothèque logicielle open source pour le calcul numérique à l'aide de graphes de flux de données. Tensorflow figure parmi les frameworks les plus utilisés et les plus adaptés pour le deep learning.

[8] https://www.tensorflow.org/

- **Keras**[9] : Écrite en Python, Keras est une API simple, flexible et puissante destinée à concevoir des réseaux de neurones artificiels et à résoudre des problèmes de machine learning, et les problèmes du deep learning. Keras a été développée dans le cadre de l'effort de recherche du projet ONEIROS (Open-ended Neuro-Electronic Intelligent Robot Operating System). Elle est soutenue par l'équipe Tensorflow de Google.
- Le code a été ensuite converti a **Tensorflow Lite**[10], qui est une bibliothèque de machine learning conçue spécifiquement pour le déploiement de modèles sur des appareils mobiles. Tensorflow Lite permet d'optimiser la taille du modèle en minimisant la taille de stockage ainsi que la taille du téléchargement et une utilisation réduite de mémoire.

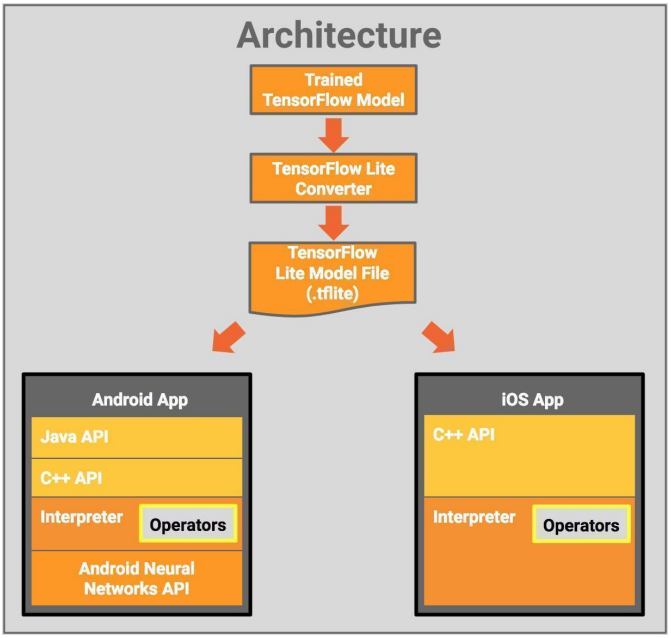


Figure 3.3 : Conversion à Tensorflow Lite

L'application a été réalisée sous l'environnement de développement Android studio avec le langage Java :

- **Android Studio** est développé par google, c'est un environnement de développement intégré (IDE) qui met à disposition des outils pour la création des applications mobiles. Il est basé sur IntelliJ IDEA. Android Studio offre des fonctionnalités qui améliorent la productivité lors de la création d'applications Android telle que l'émulateur.

[9] https://keras.io/api/
[10] TensorFlow Lite | ML pour appareils mobiles et de périphérie

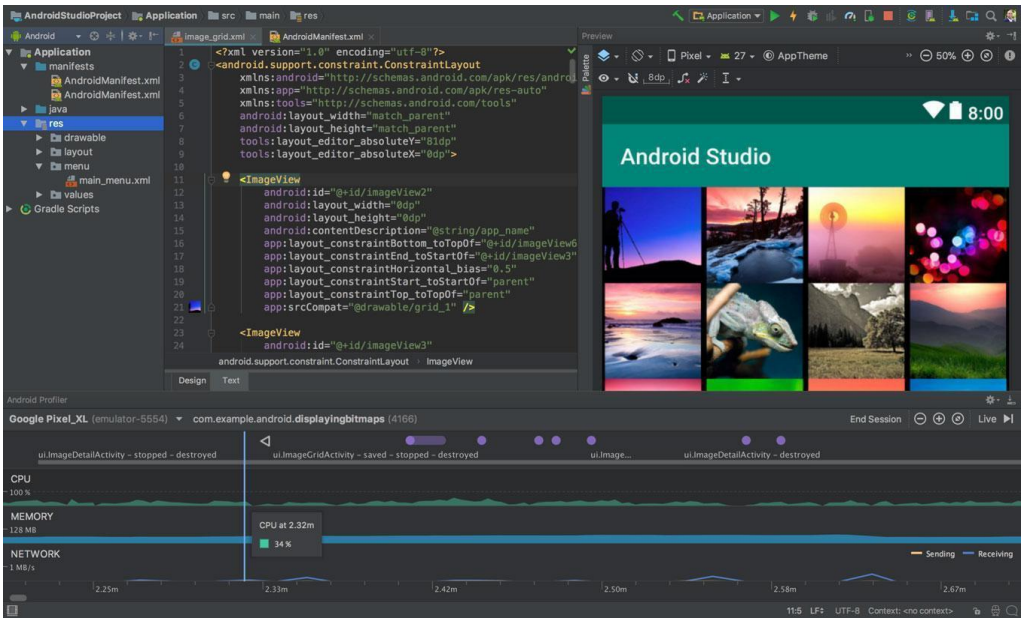


Figure 3.4 : L'environnement Android Studio

- **Java**, créé par James Gosling et Patrick Naughton, en 1995. C'est un langage de programmation orienté objet, dans lequel le développeur crée des objets, des concepts ou des idées mis en relation avec d'autres idées. Chaque objet a sa fonction et peut interagir avec les autres objets.

4.3 Exécution, Résultats

4.3.1 Fonction de compilation

Les paramètres importants de la fonction compile de keras sont les suivants : la fonction de perte, l'optimiseur et la métrique.

La fonction de perte ou loss

C'est la fonction que le modèle va utiliser pour minimiser les erreurs. Elle peut être définie par son appellation (exemple : categorical_crossentropy)

L'optimiseur ou optimizer

Il peut s'agir d'un optimiseur défini par son appellation, par exemple SGD, ou Adam de la classe Optimizer.

La métrique ou metrics

Une métrique est une fonction utilisée pour évaluer les performances du modèle. Elle est similaire à la fonction de perte, mais n'est pas utilisée dans le processus de formation. On peut

utiliser n'importe quelle fonction de perte comme métrique. Quelques métriques disponibles dans Keras :

Pour un problème de classification multi-classe ou binaire :

metrics=[‘accuracy’] L’exactitude (accuracy) est une métrique pour évaluer la performance du modèle de classification. L’exactitude calcule la fréquence à laquelle les prédictions correspondent aux labels.

$$accuracy = \frac{VP+VN}{VP+VN+FP+FN} \qquad (3.1)$$

Avec :

VP : les cas où la prédiction est positive, et où la valeur réelle est effectivement positive.

VN : les cas où la prédiction est négative, et où la valeur réelle est effectivement négative.

FP : les cas où la prédiction est positive, mais où la valeur réelle est négative.

FN : les cas où la prédiction est négative, mais où la valeur réelle est positive.

Pour un problème de régression d’erreur quadratique moyenne :

metrics=[‘mean_squared_error’’] La MSE, ou erreur quadratique moyenne, est la moyenne des carrés des erreurs, définie par la formule :

4.3.2 Bases de données

a. Pl@ntent-300k

Pl@ntnet300k[11] est la base de données de l’application française Pl@ntnet. Elle contient 306146 images de 1081 espèces construites à partir de la base de données de l'observatoire citoyen Pl@ntnet. Pl@ntnet est de 30 GO.

[11] https://zenodo.org/record/4726653#.YsRLI0XMJPY

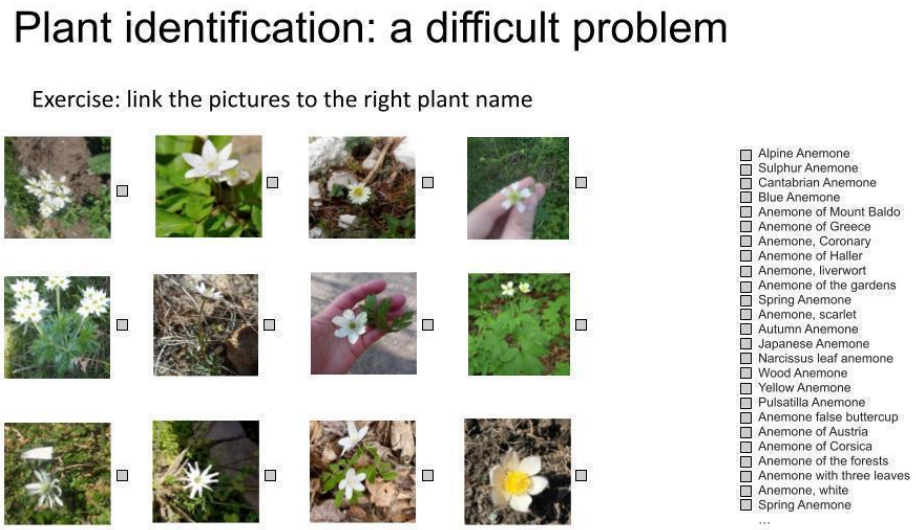


Figure 3.5 : échantillon de la base de données Pl@ntnet300k

b. Tropic20

La base de données Tropic20[12] compte 5276 images agricoles de 20 classes différentes (Acacia, Ashoka, Bambou, Banian, Absinthe chinoise, Croton, Fleur de couronne, Ervatamia, Douche dorée, Hibiscus, Palmier, Citron vert, Mangue, Tamarin de Manille, Poinsettia, Framboise bougainvillée, Sanchezia, Arbre parapluie, Jasmin antillais, Plumeria blanc) Chacune des classes de cette base contient entre 221 à 371 images. Les images sont prises dans les mêmes conditions d'éclairage, mais des parties de plantes différentes (fleurs, branches, fruits, feuilles ou l'arbre entier). De plus, certaines images comportent des arrières plans non dégagés telles que le ciel, les maisons et le sol. En voici un échantillon :

Figure 3.6 : échantillon de la base de données Tropic20

[12] Pornntiwa Pawara, Artificial Intelligence and Cognitive Engineering (ALICE) (rug.nl)

c. AgrilPlant

La base de données AgrilPlant[13] compte 3000 images agricoles de 10 classes différentes (apple, banana, grape, jackfruit, orange, papaya, persimmon, pineapple, sunflower, et tulip) collectées à partir du site Web www.flickr.com. Chacune des classes de cette base contient exactement 300 images. Les images sont prises de différentes formes de vue : la plante entière, la branche, la fleur, le fruit et la feuille. En voici un échantillon :

Figure 3.7 : échantillon de la base de données AgrilPlant

4.3.3 Techniques d'apprentissage

a. Apprentissage par transfert

L'apprentissage par transfert (domain adaptation, transfer learning en anglais) est une technique qui permet d'exploiter des modèles très profonds pré-entraînés sur de grandes bases de données, et d'en tirer profit par un transfert de connaissances acquises lors d'entraînement de ces modèles. Cette technique rend l'apprentissage plus rapide, sur les petits jeux de données, et évite le problème de sur-apprentissage. Avec cette technique les paramètres et les poids ne sont pas initialisés aléatoirement, ils auront les valeurs validées avec les modèles pré-entraînés.

L'API keras met à disposition de nombreux modèles pré-entraînés sur la grande base : ImageNet.

b. Réglage fin

Fine tuning (le réglage fin) est une approche de l'apprentissage par transfert, qui suit la formation des modèles qui ont convergé, afin d'apporter des améliorations incrémentielles. Avec fine-tuning, on modifie la sortie du modèle pour l'adapter à la nouvelle tâche, ensuite la

[13] Pornntiwa Pawara, Artificial Intelligence and Cognitive Engineering (ALICE) (rug.nl)

formation de celui-ci est relancée. On peut éventuellement débloquer/dégeler le reste du réseau et continuer la formation.

Une méthode de réglage fin consiste non seulement à remplacer les couches entièrement connectées du modèle pré-entraîné par un nouvel ensemble de couches entièrement connectées à recycler sur un jeu de données, mais à affiner tout ou partie des noyaux dans la base convolutive pré-entraînée par rétropropagation (Figure 3.8). Toutes les couches de la base convolutive peuvent être affinées ou, alternativement, certaines couches antérieures peuvent être fixées tout en affinant le reste des couches plus profondes. Ceci est motivé par l'observation que les caractéristiques de la première couche semblent plus génériques, y compris des caractéristiques telles que les bords applicables à une variété d'ensembles de données et de tâches, alors que les caractéristiques ultérieures deviennent progressivement plus spécifiques à un ensemble de données ou à une tâche particulière.

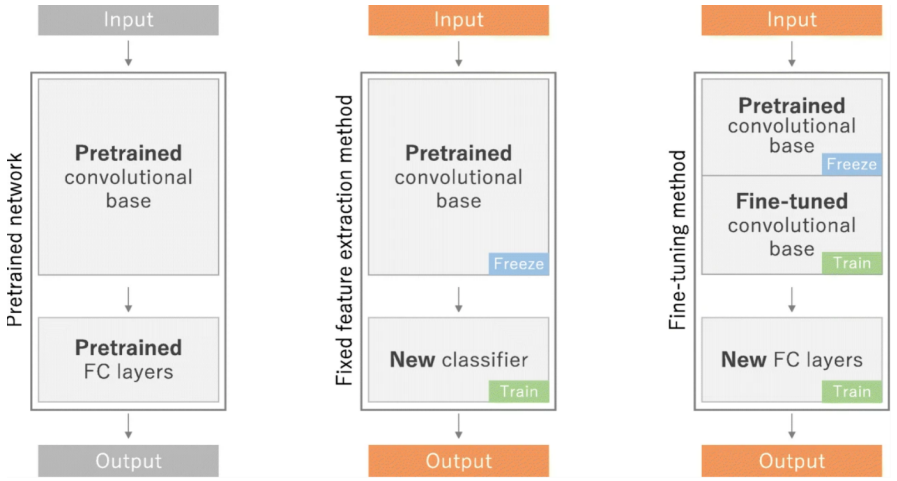


Figure 3.8 : Réglage fin [9]

c. Augmentation de données

L'augmentation de données représente l'une des astuces les plus courantes pour améliorer les performances du modèle. Cette astuce permet d'avoir beaucoup plus de données dans la base d'apprentissage. Augmenter les données c'est appliquer diverses transformations aux images de la base de données, telles que : la rotation d'image, le changement de niveaux de gris d'image, la modification de la luminosité de l'image...etc. Afin d'avoir plus de diversité dans l'ensemble d'entraînement et d'éviter les problèmes de sous-apprentissage ou de sur-apprentissage.

La règle à respecter avec cette technique pour avoir de meilleurs résultats est qu'il ne faut pas appliquer des transformations qui changeraient la bonne classe de l'image d'entrée. Par

exemple, une image qui représente le chiffre 9 ne doit pas subir une rotation de 180°, sinon il devient un 6. L'augmentation des données se produit uniquement dans la mémoire lors de l'entraînement et non pas sur le disque dur.

4.3.4 La matrice de confusion

La matrice de confusion est une mesure de performance du système. C'est une matrice qui résume les résultats de prédiction obtenu pour chaque classe d'une base de test. La matrice de confusion permet d'indiquer, pour une base de test, le nombre de prédictions correctes et incorrectes de chaque classe. La matrice doit être à diagonale dominante pour un système parfait.

4.3.5 Traitement d'image

Avant l'entraînement de réseau de neurones, les images d'entrée doivent être normalisées dans une plage [0, 1] ou [-1, 1]. Normaliser des données revient à soustraire la moyenne de chaque pixel puis à diviser le résultat par l'écart type. La normalisation des images garantit que chaque pixel a une distribution de données similaire. La distribution de ces données ressemblerait donc à une courbe de forme de cloche, centrée sur zéro. Cela rend la convergence plus rapide lors de l'entraînement du réseau.

4.3.6 Résumé des résultats

a. Expérimentation 1 : Pl@ntnet-300k

Le résultat obtenu avec la base Pl@ntnet est visualisé dans le tableau ci-dessous :

	L'ensemble d'entraînement		L'ensemble de validation		
Numéro d'époque	L'exactitude	Perte	L'exactitude	Perte	Temps d'exécution
1	3%	5.9655	3%	5.5059	4 h 32 min 38 sec
2	3%	5.3839	3%	5.4062	4 h 34 min 31 sec
3	3%	5.3398	3%	5.3966	3 h 21 min 19 sec
4	3%	5.3314	3%	5.3951	3 h 19 min 53 sec
5	3%	5.3286	3%	5.3953	3 h 16 min 19 sec

Tableau 3-1 : Résultat d'apprentissage sur la base de données Pl@ntnet

Discussion du résultat de la base Pl@ntnet :

La base Pl@ntnet300k a été compilée par la méthode de l'apprentissage par transfert. Sur 5 époques d'entraînement elle nous a pris 19 heures 30 minutes et présente les difficultés suivantes :

- Le déséquilibre des classes dans cette base ; certaines espèces représentent la plupart des images avec plus de 1000 images, alors que d'autres espèces n'ont que 2 images.
- La diversité présente dans l'ensemble de données de Pl@ntnet ; nombreuses espèces sont visuellement similaires, ce qui rend l'identification difficile même pour l'œil expert.
- La taille gigantesque de la base Pl@ntnet300k fait de son exécution un problème majeur. L'exécution de cette base sur la machine locale nous présentait un problème persistant : Noyau mort (Kernel died). On a donc essayé d'exécuter cette base sous le

service Azure Machine Learning, qui est un service pensé pour l'entraînement des modèles de deep learning sur de grande quantité de données. Les résultats constatés sur 5 époques nous montrent une faible amélioration du la fonction perte d'époque à l'autre. Ce qui est un signe que cette base aura besoin de beaucoup plus de temps et d'époques. Cependant, le service Azure Machine Learning n'est pas 100% gratuit, de ce fait on a changé de base de données pour la suite de nos expérimentations.

b. Expérimentation 2 : L'apprentissage par transfert

Pour cette expérimentation, on a appliqué la technique de l'apprentissage par transfert sur les deux bases de données : Tropic20 et AgrilPlant. Les résultats sont visualisés ci-dessous :

1.Le taux d'exactitude et la perte pour chaque ensemble :

	L'ensemble d'entrainement		L'ensemble de validation		L'ensemble de test		Le Temps d'entrainement
	Exactitude	Perte	Exactitude	Perte	Exactitude	Perte	
Tropic20	76%	0.8014	68%	1.0307	68%	1.0391	29 min 50 sec
AgrilPlant	81%	0.5967	73%	0.7688	76%	0.7478	18 min 45 sec

Tableau 3-2 : Résultats d'apprentissage par transfert

2. Les courbes d'apprentissage

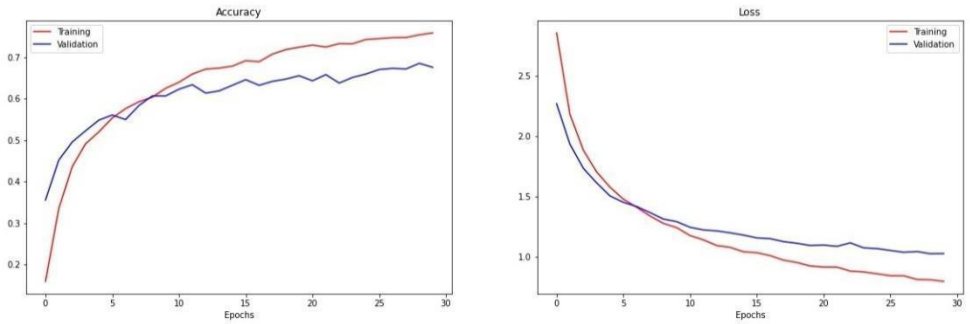


Figure 3.9 : Courbes d'apprentissage - apprentissage par transfert - Tropic20

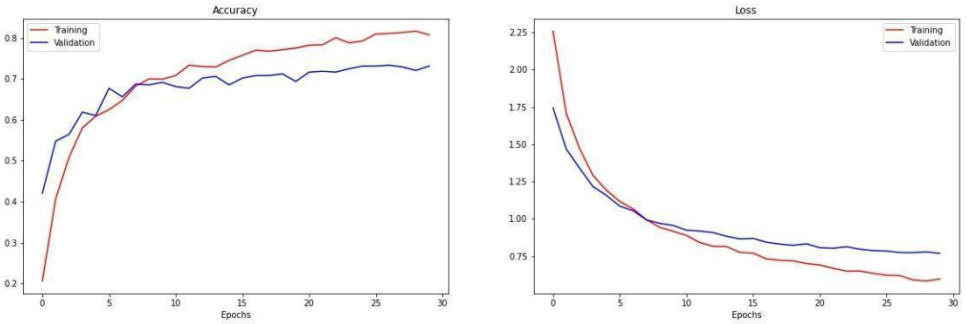


Figure 3.10 : Courbes d'apprentissage - apprentissage par transfert - AgrilPlant

3.La matrice de confusion

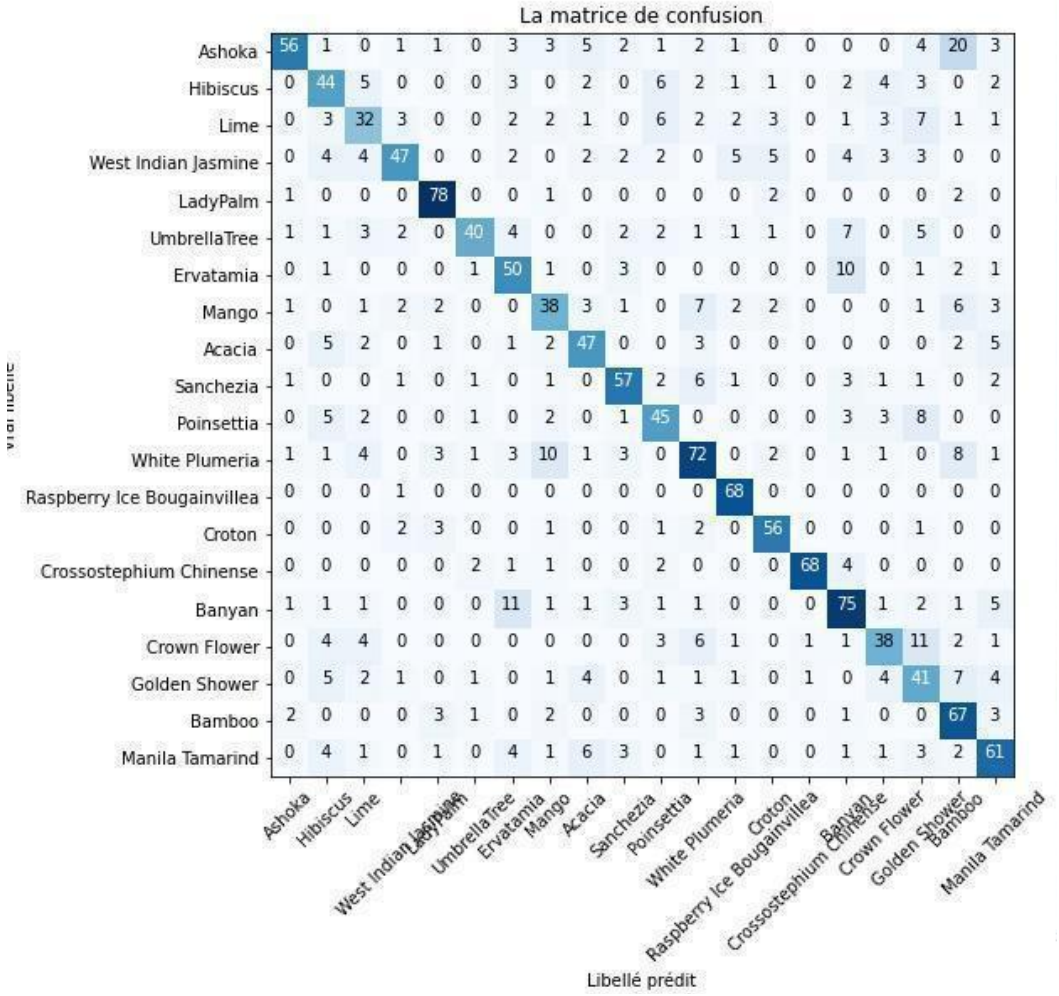


Figure 3.11 : matrice de confusion - apprentissage par transfert - Tropic20

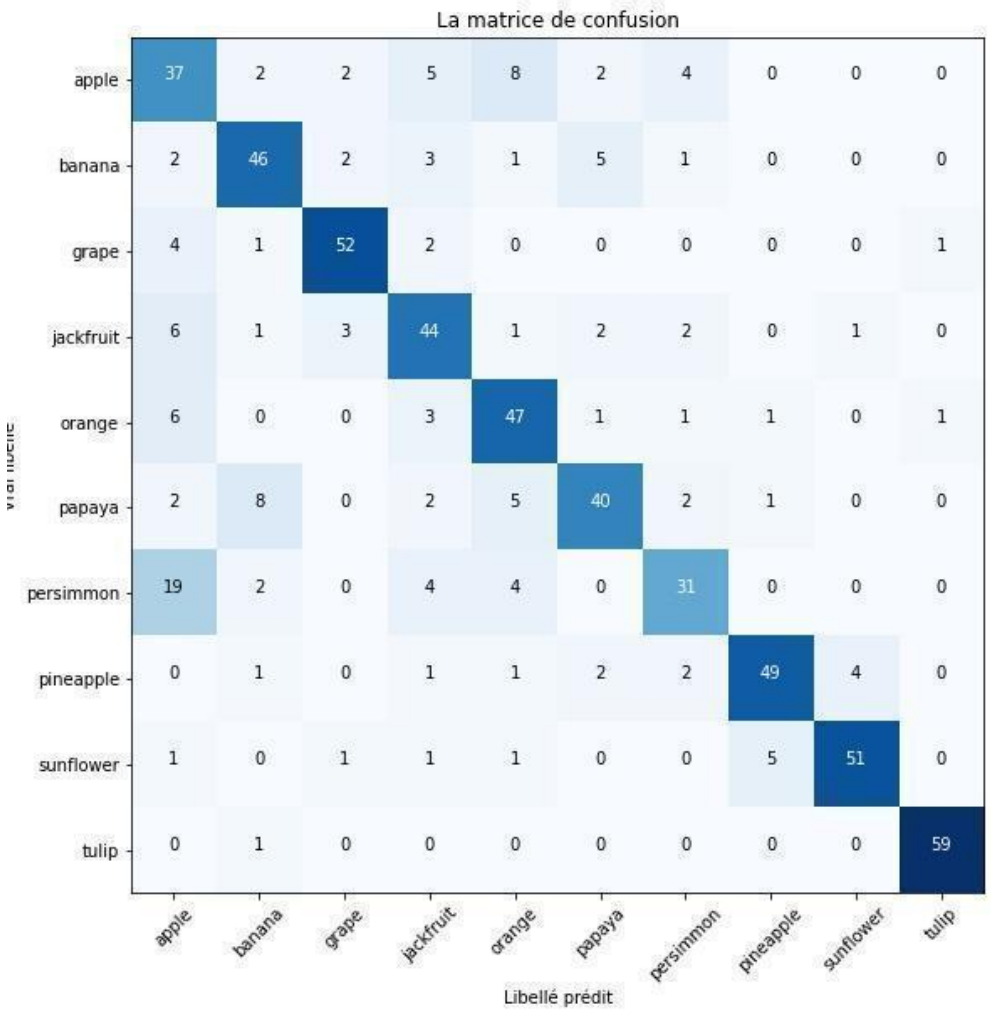


Figure 3.12 : matrice de confusion - apprentissage par transfert - AgrilPlant

4.Interprétation des résultats :

Pour les deux bases de données, les courbes d'apprentissages (Figure 3.9, Figure 3.10) montrent que la courbe d'entrainement s'éloigne petit à petit de la courbe de validation au fil des époques ce qui peut conduire à une situation de sur-apprentissage. Des méthodes de régularisation peuvent être envisagées pour régler ce problème (tel que l'augmentation de données, ou le dropout. Détaillés dans le chapitre 2).

La matrice de confusion pour la base Tropic20 indique que le meilleur score est pout la classe Lady Palm. Et la classe tulip pour la base AgrilPlant.

c. Expérimentation 3 : Le réglage fin

Par cette expérimentation on a utilisé la technique de réglage fin sur les deux bases de données. Les résultats obtenus sont visualisés ci-dessous :

1.Le taux d'exactitude et la perte pour chaque ensemble :

	L'ensemble d'entrainement		L'ensemble de validation		L'ensemble de test		Le Temps d'entrainement
	Exactitude	Perte	Exactitude	Perte	Exactitude	Perte	
Tropic20	99%	0.0471	78%	0.8773	78%	0.7943	59 min 11 sec
AgrilPlant	98%	0.0655	82%	0.6695	81%	0.6786	37 min 37 sec

Tableau 3-3 : Résultats réglage fin

2. Les courbes d'apprentissage

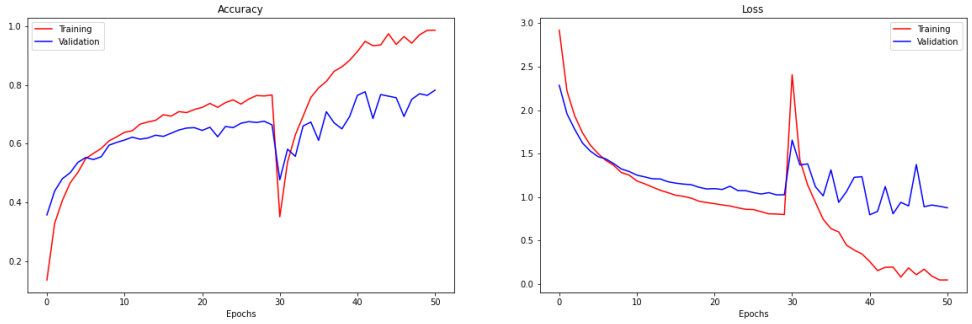


Figure 3.13 : Courbes d'apprentissage - Réglage fin - Tropic20

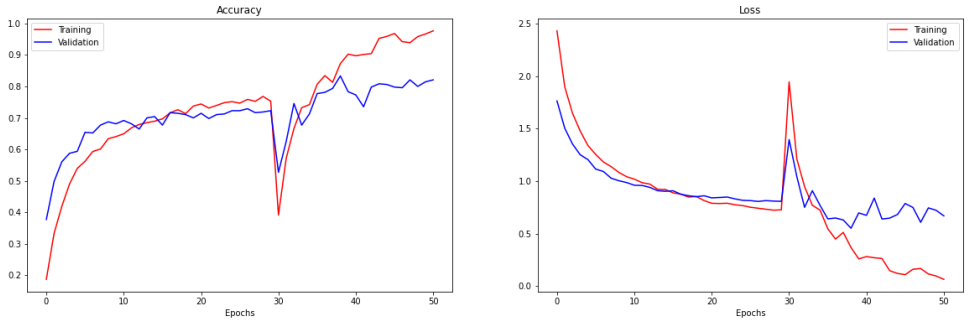


Figure 3.14 : Courbes d'apprentissage - Réglage fin - AgrilPlant

3.Matrices de confusion

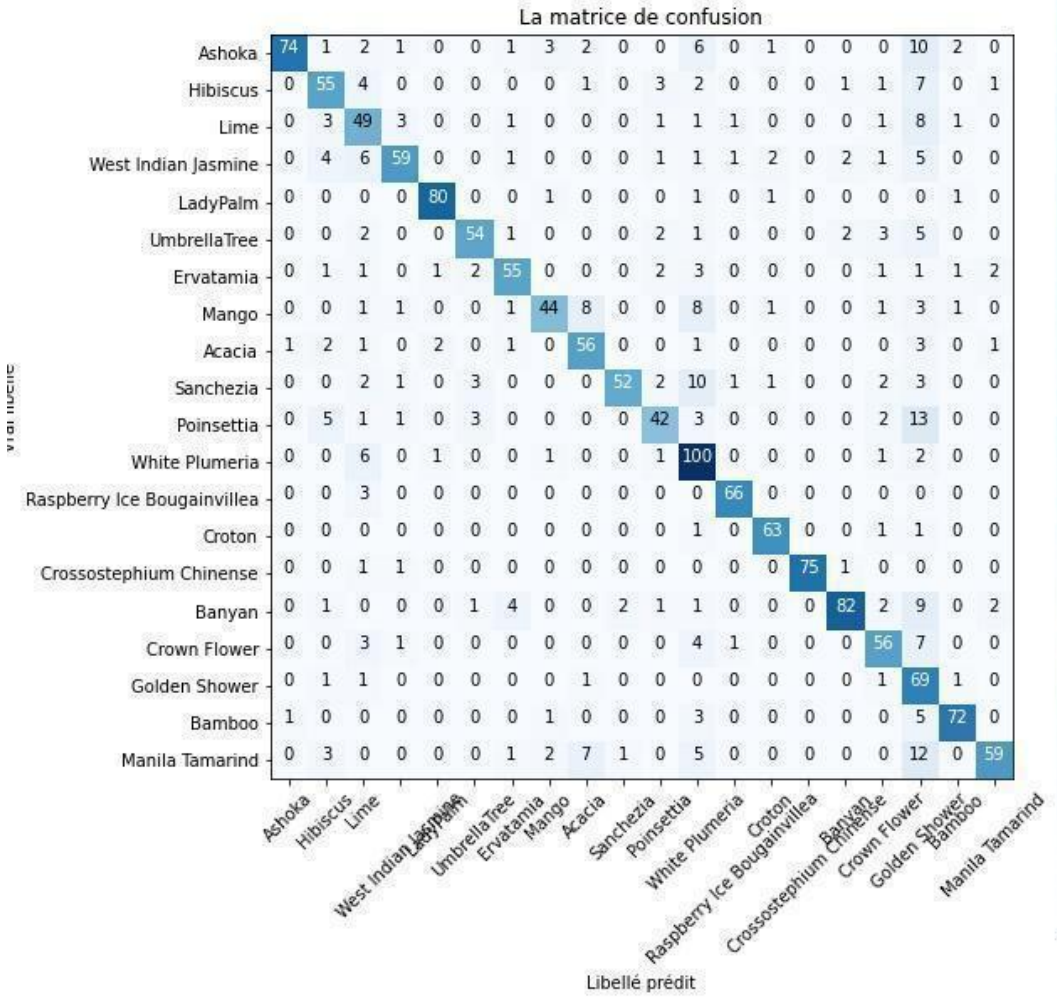


Figure 3.15 : Matrice de confusion - Réglage fin - Tropic20

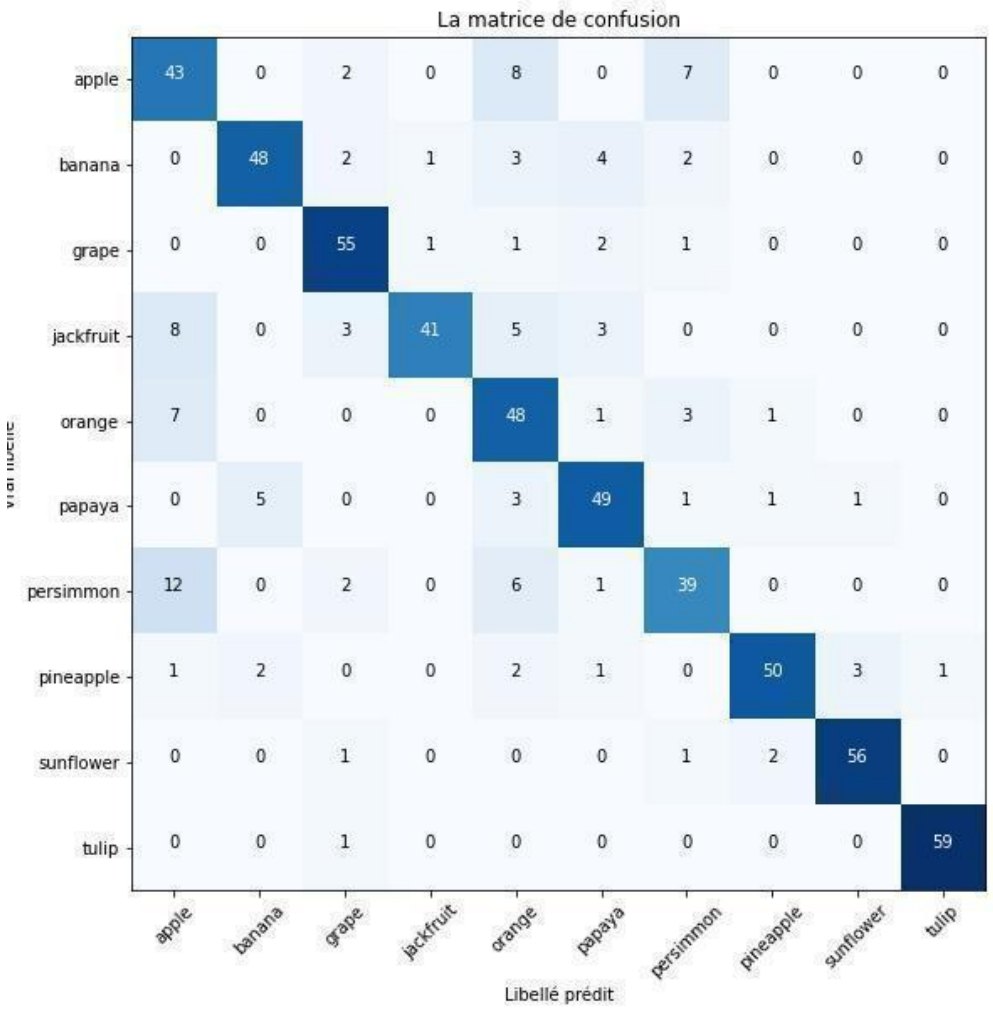


Figure 3.16 : Matrice de confusion - Réglage fin - AgrilPlant

4.Interprétation de résultats

Les matrices de confusion résultant de la technique de réglage fin sont en diagonale dominante. Les courbes d'apprentissage indiquent que le modèle est en sur-apprentissage ; étant donné que la courbe d'exactitude d'entraînement augmente très vite, sur la deuxième phase contrairement à la courbe qui représente l'exactitude l'ensemble de validation.

d. Expérimentation 4 : L'augmentation de données

Tropic20 et AgrilPlant ne sont pas des bases de données volumineuses. On espère avec la technique d'augmentation de données (détaillé dans le chapitre 3) avoir de meilleurs résultats.

1.Le taux d'exactitude et la perte pour chaque ensemble :

	L'ensemble d'entrainement		L'ensemble de validation		L'ensemble de test		Le Temps d' entrainement
	Exactitude	Perte	Exactitude	Perte	Exactitude	Perte	
Tropic20	96%	0.1589	96%	0.158	91%	0.3099	41 min 39 sec
AgrilPlant	97%	0.0871	98%	0.0872	96%	0.093	44 min 35 sec

Tableau 3-4 : Résultats d'augmentation de données

2. Les courbes d'apprentissage

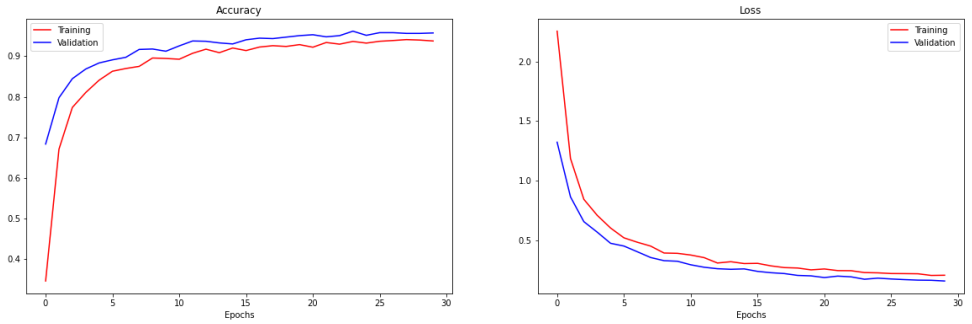


Figure 3.17 : Courbes d'apprentissage -augmentation de données - Tropic20

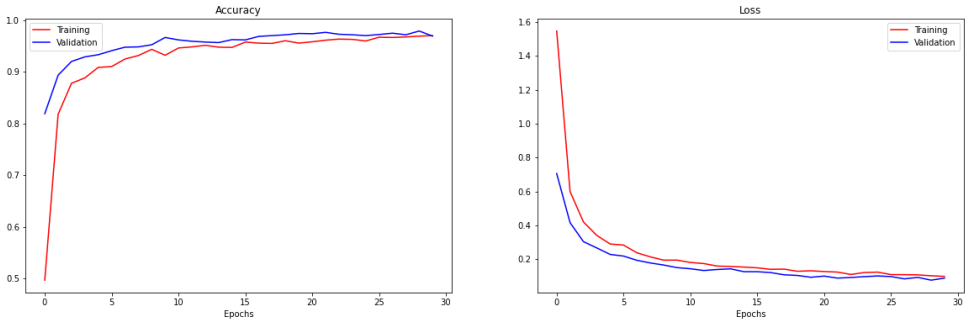


Figure 3.18 : Courbes d'apprentissage -augmentation de données - -AgrilPlant

3.Matrices de confusion

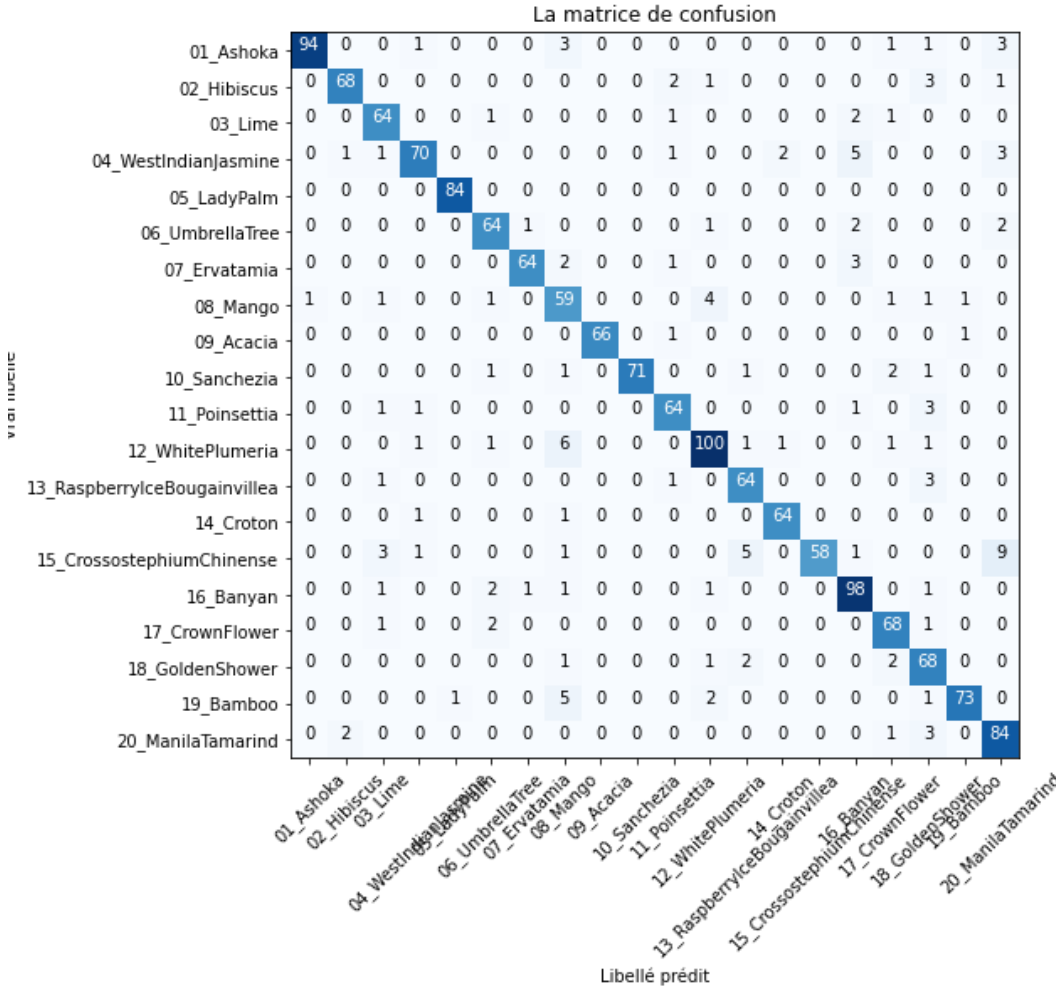


Figure 3.19 : Matrice de confusion - augmentation de données - Tropic20

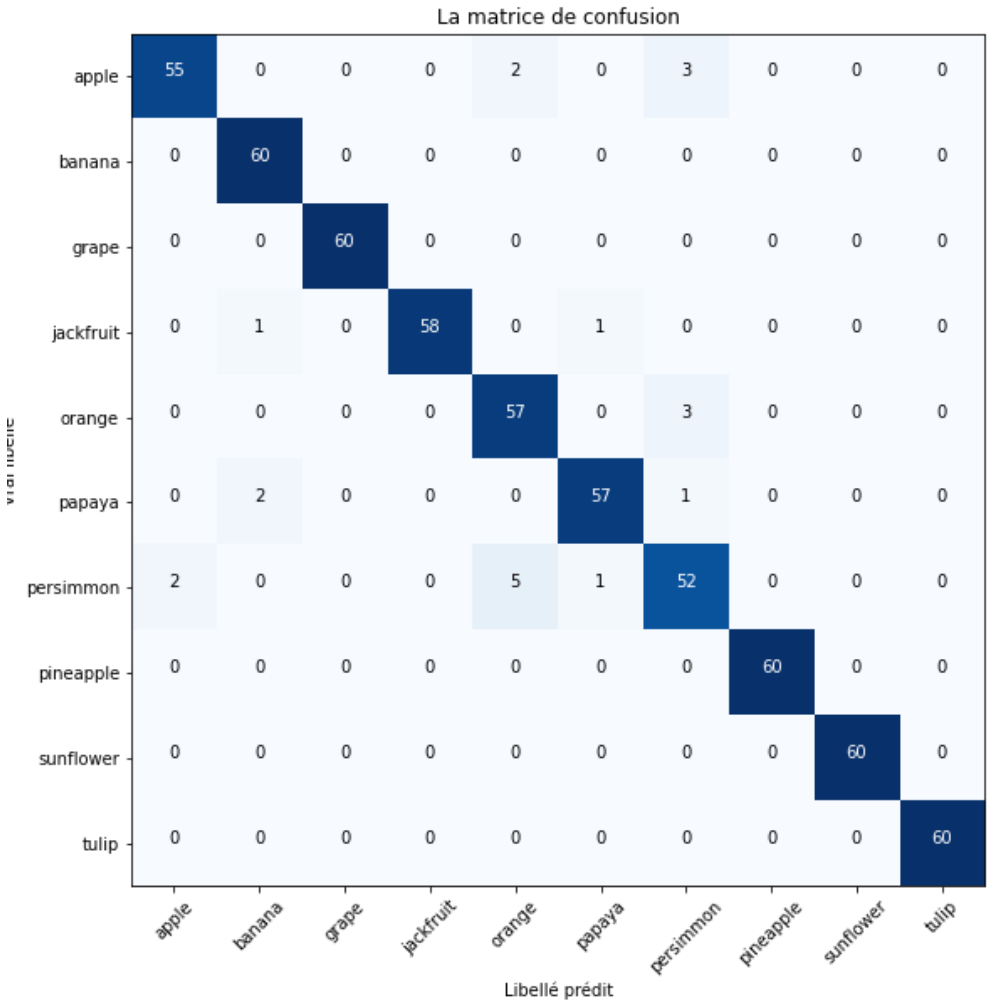


Figure 3.20 : Matrice de confusion - augmentation de données - AgrilPlant

4.Interprétation des résultats

Comme on peut le constater, l'augmentation de données nous a donné les meilleurs résultats. Avec jusqu'à 92% et 96% d'exactitude sur l'ensemble de test, sur la base Tropic20 et AgrilPlant respectivement.

D'après les matrices de confusion, le modèle est bon pour toutes les classes des deux bases de données.

e. Expérimentation 5 : Le réglage fin et l'augmentation de données

Maintenant nous allons associer les techniques d'augmentation de données et de réglage fi, dans le but d'augmenter les performances de notre modèle sur les deux bases de données. Les résultats qu'on a obtenus sont visualisés ci-dessous :

1.Le taux d'exactitude et la parte pour chaque ensemble :

	L'ensemble d'entrainement		L'ensemble de validation		L'ensemble de test		Le Temps d'entrainement
	Exactitude	Perte	Exactitude	Perte	Exactitude	Perte	
Tropic20	99.82%	0.0110	99.63%	0.0142	97%	0.1297	1 h 19 min 5 sec
AgrilPlant	99.22%	0.017	99.22%	0. 0184	98%	0.0702	1 h 16 min 28 sec

Tableau 3-5 : Résultats de réglage fin et augmentation de données

2. Les courbes d'apprentissage

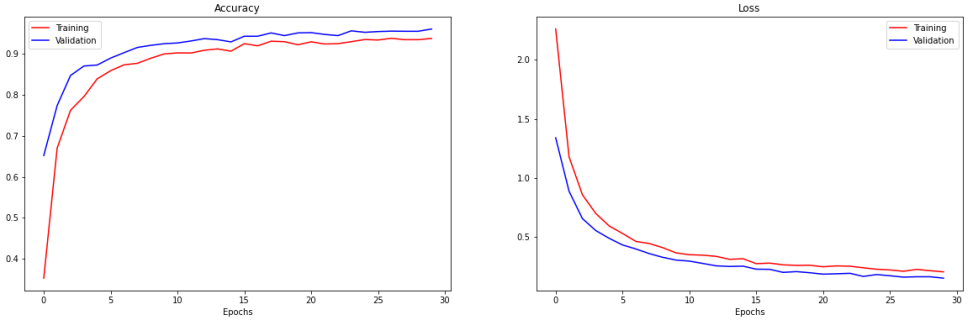


Figure 3.21 : Courbes d'apprentissage - réglage fin et augmentation de données - Tropic20

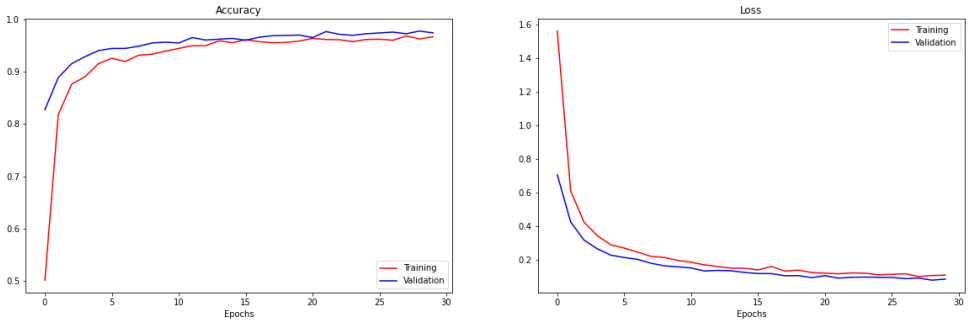


Figure 3.22 : Courbes d'apprentissage - réglage fin et augmentation de données - AgrilPlant

3.Matrices de confusion

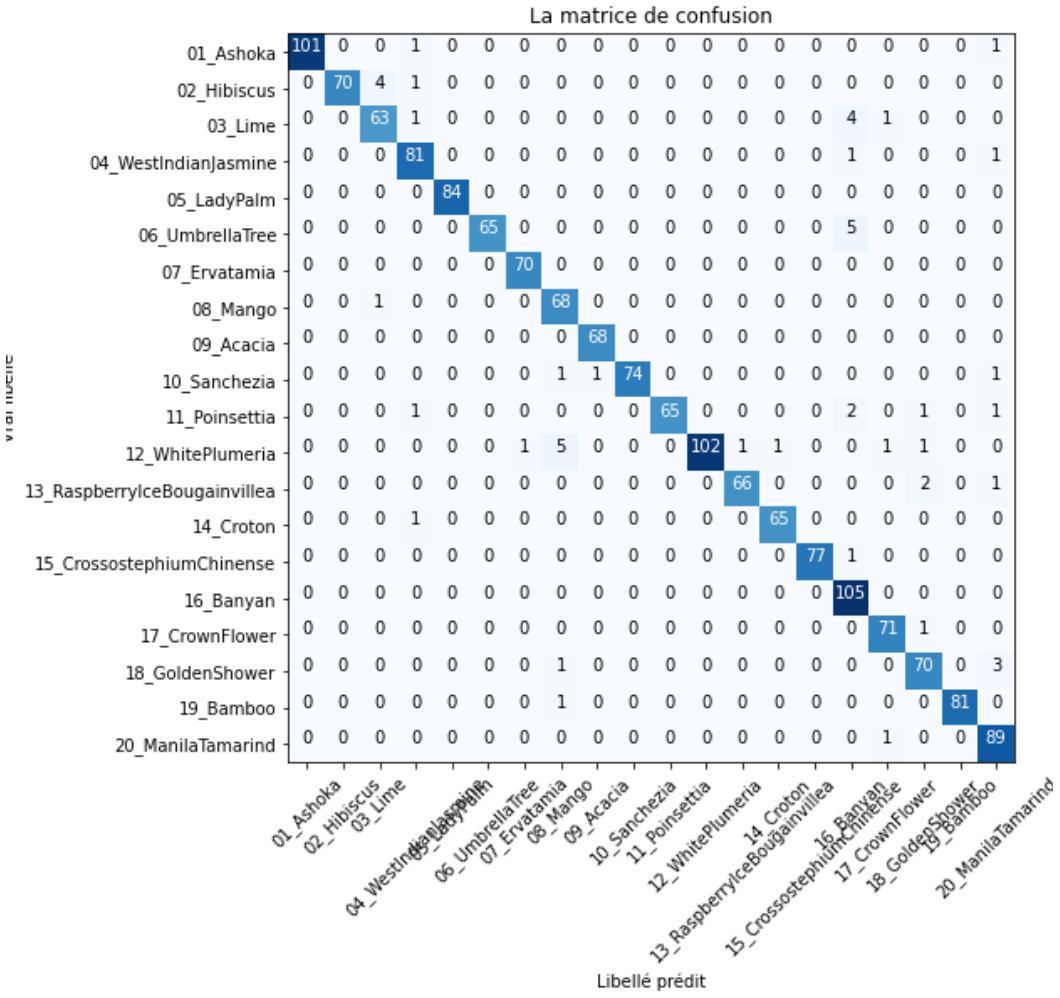


Figure 3.23 : Matrice de confusion - Augmentation de données et réglage fin - Tropic20

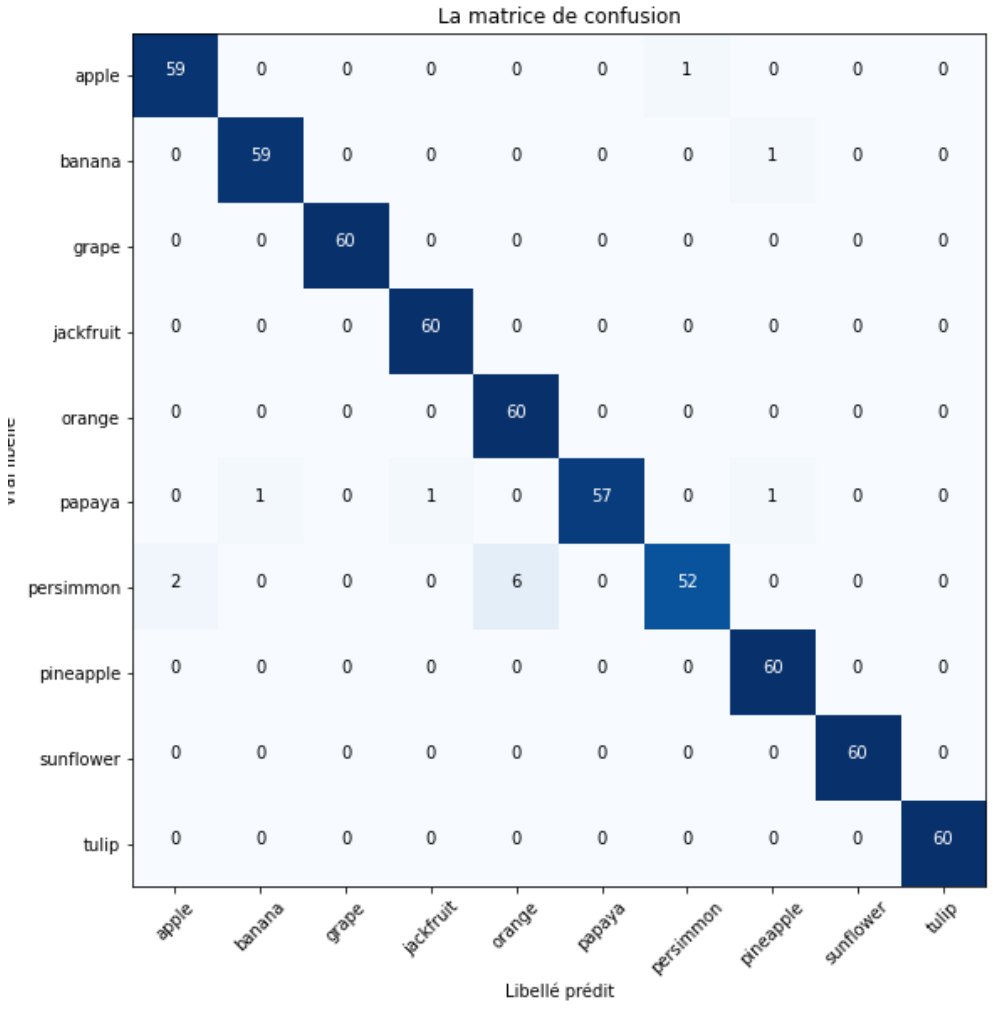


Figure 3.24 : Matrice de confusion - Augmentation de données et réglage fin - AgrilPlant

4.Interprétation des résultats :

Comme on peut le voir sur les résultats de matrice de confusion, le réglage fin avec l'augmentation de données a permis d'améliorer les résultats.

f. Discussion et analyse des résultats

Nous avons testé les performances du modèle en représentant les courbes d'apprentissage et les matrices de confusion.

Nous constatons que les matrices de confusion obtenues sont à diagonale dominante et l'augmentation de données a permis d'améliorer les résultats.

4.4 Résultats de prédiction

Une fois que le réseau de neurones a été entraîné on peut enfin faire des prédictions sur des données nouvelles que le réseau n'a jamais vues auparavant.

Prédictions	L'image	Le résultat retourné par le modèle
Avec une image nette et arrière-plan dégager		apple = 8.5% banana = 8.5% grape = 8.5% jackfruit = 8.5% orange = 8.5% papaya = 8.5% persimmon = 8.5% pineapple = 8.5% sunflower = 8.5% **tulip = 23.2%**
Avec une image qui comporte le bruit flou		apple = 8.5% banana = 8.5% grape = 8.5% jackfruit = 8.5% orange = 8.5% papaya = 8.5% persimmon = 8.5% pineapple = 8.5% **sunflower = 23.2%** tulip = 8.5%
Avec une image nette et arrière-plan non dégagé		apple = 8.5% banana = 8.5% grape = 8.5% jackfruit = 8.5% orange = 8.5% papaya = 8.5% persimmon = 8.5% **pineapple = 23.19%** sunflower = 8.5% tulip = 8.5%

Prédictions	L'image	Le résultat retourné par le modèle
Avec une image nette et arrière-plan non dégagé		apple = 8.5% banana = 8.5% grape = 8.5% jackfruit = 8.5% **orange = 23.18%** papaya = 8.5% persimmon = 8.5% pineapple = 8.5% sunflower = 8.5% tulip = 8.5%
Avec une image de vision loin et arrière-plan non dégagé		apple = 8.5% banana = 8.6% grape = 8.5% **jackfruit = 23.11%** orange = 8.5% papaya = 8.5% persimmon = 8.5% pineapple = 8.5% sunflower = 8.5% tulip = 8.5%

Discussion des résultats de prédiction

Plusieurs images ont été passées à notre modèle, et pour chaque image notre modèle reconnaît la classe correspondante. Même avec des images prises dans des conditions non favorables telles que : des images qui comportent le bruit flou, des images de vision lointaine ou des images avec un arrière-plan non dégagé.

Il est à noter que notre modèle se trompe parfois ; il ne peut pas reconnaître des espèces en dehors des 10 classes d'AgrilPant.

Si les CNN sont une référence dans le domaine du traitement numérique des images et de la vision par ordinateur, ils ont tout de même quelques inconvénients :

1. Les CNN (au même titre que les réseaux de neurones, de manière générale) sont très gourmands en termes de données. En effet, pour qu'un réseau de neurones soit capable de reconnaître un seul objet, il a besoin de plusieurs images prises différemment l'une de l'autre de ce même objet, pour pouvoir faire une généralisation. Les CNN sont mauvais pour gérer la rotation et l'invariance d'échelle sans augmentation explicite des données. Cela fait d'eux de grands consommateurs de données.
2. Les CNN ont aussi besoin de beaucoup de temps pour l'entraînement, surtout si la base de données est gigantesque.
3. Les CNN exigent un matériel puissant pour l'entraînement comme un GPU.

4.5 Déploiement de l'application

Nous allons à présent déployer notre modèle, nous avons procédé ainsi :

1. Construction de l'interface de l'application Nabet-ID :

Suivants les règles de UI/UX design, on a choisi de créer deux interfaces pour notre application : La première est l'interface principale de l'application Nabet-ID (Figure 4.21), elle contient deux boutons un pour la prise de photo en directe (désigné par : CAMERA) l'autre permet de récupérer une photo depuis la galerie du mobile de l'utilisateur (désigné par : GALERIE). Le résultat sera retourné sur la deuxième interface (Figure 4.22).

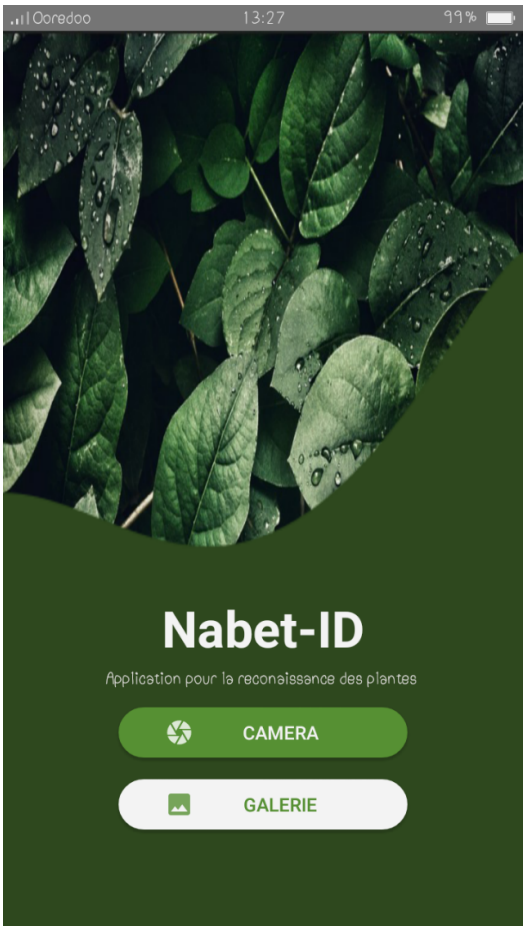


Figure 3.25 : Interface 1 de l'application Nabet-ID

Figure 3.26 : Interface 2 de l'application Nabet-ID

2. Chargement du modèle :

Nous avons ensuite chargé notre modèle (en .tflite) dans Android Studio, une fois que notre modèle s'est chargé, nous avons intégré un code qui nous permet d'afficher le résultat retourné à partir du modèle.

4.6 Conclusion

La base de données AgrilPlant avec augmentation de données et réglage fin nous a donné satisfaction. L'implémentation nous a permis de mettre en évidence les difficultés liées à l'identification des plantes par CNN, comme l'impossibilité d'exécuter la base de données Pl@ntent à cause de sa taille, ce qui nous a obligé d'utiliser d'autres base de données moins volumineuses.

Conclusion Générale et Perspectives

Dans ce travail, nous avons développé l'application Nabet-ID avec l'architecture de réseau de neurones MobileNetV2, sur la base de données AgrilPlant. Ayant obtenu de bons résultats à l'aide de techniques de réglage fin et d'augmentation de données, nous avons déployé l'application avec Java sous l'environnement de développement Android Studio.

Il serait intéressant par la suite d'adapter Nabet-ID à une base de données algérienne, locale puis nationale. Elle pourra également être développée pour contribuer aux tâches de classification, positionnement, dénombrement statistique, cartographie botanique ... etc. Une autre utilisation de cette application est l'exploitation en phytothérapie, par l'identification des plantes médicinales avec leur principes, mode d'emploi et l'option de développement d'un descriptif spécialisé plus détaillé et précis de ces plantes. Une autre voie d'application est la détermination d'éventuelles phytopathologies et leurs traitements appropriés dans le domaine de l'agronomie. Elle pourra enfin aider à l'implantation de nouvelles variétés agricoles, conformément aux disponibilités des ressources en eau des diverses régions.

L'application Nabet-ID constitue donc une opportunité économique et un créneau stratégique, d'exploitation et d'exportation, d'emploi et de formation, au niveau de notre zone géographique produisant notre propre base de données nationale et/ou régionale. Cette application sera indépendante et rivalisera avec des applications similaires existantes au niveau des autres pays.

En conclusion, de très nombreux domaines et champs d'activités pourront être revisités et rafraichis avec les apports de l'application Nabet-ID, qui se veut un outil stratégique à incidences multiples.

Annexe 3: Code source

AgrilPlant - Augmentation de données - Réglage fin

Importation de bibliothèques:

In [ ]:
```
import PIL

import time

import numpy as np

import os, warnings

import matplotlib.pyplot as plt
from matplotlib import gridspec

import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras.models import Sequential
from tensorflow.keras import Sequential, optimizers, layers, Model
from tensorflow.keras.preprocessing.image import ImageDataGenerator
```

Chargement et prétraitement de base AgrilPlant

In [ ]:
```
train_dir = "C:\\Users\\TUF GAMING\\Desktop\\Datasets\\Mabase1\\train"
validation_dir = "C:\\Users\\TUF GAMING\\Desktop\\Datasets\\Mabase1\\validation"
test_dir = "C:\\Users\\TUF GAMING\\Desktop\\Datasets\\Mabase1\\test"
```

In [ ]:
```
batch_size_train = 64
batch_size_test = 1
input_shape = (224,224,3)
num_classes = 10
epochs_phase1 = 30
```

In [ ]:
```
train_datagen = ImageDataGenerator(rescale=1./255,
                                   zoom_range=0.2,
                                       rotation_range=40,
                                           width_shift_range=0.2,
                                               height_shift_range=0.2)



train_generator = train_datagen.flow_from_directory(train_dir,
                                                    seed=42,
                                                    shuffle=True,
                                                    color_mode="rgb",
                                                    target_size=(224, 224),
                                                    class_mode="categorical",
                                                    batch_size=batch_size_train)

validation_datagen = ImageDataGenerator(rescale=1./255,
                                        zoom_range=0.2,
                                            rotation_range=40,
                                                width_shift_range=0.2,
                                                    height_shift_range=0.2)



validation_generator = validation_datagen.flow_from_directory(train_dir,
                                                              seed=42,
                                                              shuffle=True,
                                                              color_mode="rgb",
                                                              target_size=(224, 224),
                                                              class_mode="categorical",
                                                              batch_size=batch_size_train)

test_datagen = ImageDataGenerator(rescale=1./255)

test_generator = test_datagen.flow_from_directory(test_dir,
                                                  target_size=(224, 224),
                                                  color_mode="rgb",
                                                  batch_size=1,
```

```
                                                class_mode="categorical",
                                                shuffle=False,
                                                seed=42)
```

Téléchargement et préparation du modèle

In [ ]:
```
# Téléchargement du modèle MobileNetV2 :
base_model = keras.applications.MobileNetV2(input_shape = input_shape,
                                            include_top = False,
                                            weights = "imagenet")
```

In [ ]:
```
#L'apprentissage par transfert
#Pour empêcher l'entrainement :
for layer in base_model.layers:
    layer.trainable = False
```

In [ ]:
```
#Définitoin du modèle
inputs = keras.Input(shape = input_shape)
x = base_model(inputs,
               training = False)
x = tf.keras.layers.GlobalAveragePooling2D()(x)
x = tf.keras.layers.Dropout(0.2)(x)
x = tf.keras.layers.Dense(num_classes,
                          activation="softmax")(x)
model = keras.Model(inputs = inputs,
                    outputs = x)
```

In [ ]:
```
# L'affichage de la structure du modèle :
model.summary()
```

In [ ]:
```
len(model.trainable_variables)
```

La configuration réseau de neurones

In [ ]:
```
# Compilation : Définir comment entraîner le modèle
model.compile(optimizer = tf.keras.optimizers.Adam(),
              loss = tf.keras.losses.CategoricalCrossentropy(),
              metrics=['accuracy'])
```

In [ ]:
```
# Entrainement:
curr_time = time.time()
t1=time.time()
STEP_SIZE_TRAIN=train_generator.n//train_generator.batch_size
history =  model.fit(train_generator,
                     validation_data = validation_generator,
                    steps_per_epoch=STEP_SIZE_TRAIN,
                    epochs=epochs_phase1)
t2=time.time()
print("Le  temps d'entraînement est : ",t2-t1)
```

Réglage fin

In [ ]:
```
# Dégeler les couches supérieures du modèle
base_model.trainable = True
```

In [ ]:
```
# Combien de couches dans base model
print("Le nombre de couche de base model est : ", len(base_model.layers))

# Fine-tune À partir de:
fine_tune_at = 100

# Geler toutes les couches avant la couche `fine_tune_at`
for layer in base_model.layers[:fine_tune_at]:
    layer.trainable = False
```

In [ ]:
```
# Re-compiler
model.compile(optimizer = tf.keras.optimizers.Adam(learning_rate=0.001/10),
```

```
              loss = tf.keras.losses.CategoricalCrossentropy(),
              metrics=['accuracy'])
```

In [ ]:
```
len(model.trainable_variables)
```

In [ ]:
```
# Continuité de la formation du modèle :
fine_tune_epochs = 20
total_epochs =  epochs_phase1 + fine_tune_epochs

curr_time = time.time()
t1=time.time()

history_fine = model.fit(train_generator,
                         validation_data = validation_generator,
                         initial_epoch=history.epoch[-1],
                         epochs = total_epochs)
t2=time.time()
print("Le  temps d'entraînement est : ",t2-t1)
```

Les Performances du modèle

1.Evaluation du modèle sur les trois ensembles

In [ ]:
```
print(">>>>>>>>>> Evaluation sur l'ensemble d'entrainement <<<<<<<<<<")
result = model.evaluate(train_generator)
dict(zip(model.metrics_names, result))
```

In [ ]:
```
print(">>>>>>>>>> Evaluation sur l'ensemble de validation <<<<<<<<<<")
result = model.evaluate(validation_generator)
dict(zip(model.metrics_names, result))
```

In [ ]:
```
print(">>>>>>>>>> Evaluation sur l'ensemble de test <<<<<<<<<<")
result = model.evaluate(test_generator)
dict(zip(model.metrics_names, result))
```

2. Les courbes d'apprentissage

In [ ]:
```
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']

loss = history.history['loss']
val_loss = history.history['val_loss']

epochs = range(len(loss))

plt.figure(figsize=(20,6))
plt.subplot(1, 2, 1)
plt.plot(epochs,acc,c="red",label="Training")
plt.plot(epochs,val_acc,c="blue",label="Validation")
plt.xlabel("Epochs")
plt.title('Accuracy')
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(epochs,loss,c="red",label="Training")
plt.plot(epochs,val_loss,c="blue",label="Validation")
plt.xlabel("Epochs")
plt.title('Loss')
plt.legend()

plt.savefig("CourbesApprentissageAgrilPlantAvecAugmentationDonnes.png")
```

3. La Matrice de confusion :

In [ ]:
```
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
from sklearn.metrics import classification_report, confusion_matrix
import itertools
```

In [ ]:
```
# Préparation de données passées en paramètre :
```

```
target_names = []
for key in test_generator.class_indices:
    target_names.append(key)

Y_pred = model.predict(test_generator)
y_pred = np.argmax(Y_pred, axis=1)
```

In [ ]:

```
# Tracer la matrice de confusion
def plot_confusion_matrix(cm, classes, normalize=False, title='La matrice de confusion', cmap=plt.cm.Blues):
    plt.figure(figsize=(10,10))
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)
    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
        cm = np.around(cm, decimals=2)
        cm[np.isnan(cm)] = 0.0
        print("Matrice de confusion normalisée")
    else:
        print('Matrice de confusion, sans normalisation')

    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, cm[i, j], horizontalalignment="center", color="white" if cm[i, j] > thresh else "black")
    plt.tight_layout()
    plt.ylabel('Vrai libellé')
    plt.xlabel('Libellé prédit')
    plt.savefig("cm.png")
```

In [ ]:

```
#L'appel à la fonction confusion_matrix :
# cas 1 :  deux paramètres : test_dataset.classes (labels réels), y_pred
print('La matrice de cofusion')
cm = confusion_matrix(test_generator.classes, y_pred)
plot_confusion_matrix(cm, target_names)

# Le rapport de classification et l'affichage de la mc :
print('Le rapport de classification')
print(classification_report(test_generator.classes, y_pred, target_names=target_names))
```

Enregistrement du modèle

In [ ]:

```
# L'enregistremnent du modèle et des poids avec keras :
from keras.models import model_from_json
from keras.models import load_model

# 1. Enregistrement du modèle :
model.save('24_model_AgrilPlant_dataAug_fintuning.h5') #en h5
model_json = model.to_json()
with open("24__model_AgrilPlant_dataAug_fintuning.json", "w") as json_file:
    json_file.write(model_json) #en json

# 2. Enregisrement des poids :
model.save_weights("28_model_AgrilPlant_dataAug_fintuning.h5")
```

La convertion du modèle Keras en Tensorflow Lite

In [ ]:

```
converter = tf.lite.TFLiteConverter.from_keras_model(model)
tflite_float_model = converter.convert()
```

In [ ]:

```
# Affichage de la taille du modèle en KBs.
float_model_size = len(tflite_float_model) / 1024
print('La taille du modèle = %d KBs.' % float_model_size)
```

Quantification

In [ ]:

```
# Re-convertir le modèle en TensorflowF Lite avec la technique de quantification :
converter.optimizations = [tf.lite.Optimize.DEFAULT]
tflite_quantized_model = converter.convert()
```

In [ ]:
```
# La taille du modèle devient :
quantized_model_size = len(tflite_quantized_model) / 1024
print('Après quantification')
print('La taille du model = %d KBs,' % quantized_model_size)
print('qui représente environ %d%% de la taille du modèle initiale.'\
      % (quantized_model_size * 100 / float_model_size))
```

Enregistrement et téléchargement du modèle :

In [ ]:
```
# Enregistrement :
f = open('24__model_AgrilPlant_dataAug_fintuning.tflite', "wb")
f.write(tflite_quantized_model)
f.close()
```

Bibliographie

[1] N. Toulon, Deep Learning et Agriculture, Chaire AgroTIC, 2018.

[2] M. P. J. W. Jaak Pärtel, «Plant image identification application demonstrates high accuracy in Northern Europe,» vol. Volume 13, n° %1Issue 4, 2021.

[3] «Près de 8,7 millions d'espèces vivantes peuplent la Terre,» Le Monde, [En ligne]. Available: https://www.lemonde.fr/planete/article/2011/08/23/pres-de-8-7-millions-d-especes-vivantes-peuplent-la-terre_1562713_3244.html. [Accès le 06 2022].

[4] H. G. Jones, «What plant is that? Tests of automated image recognition apps for plant identification on plants from the British flora,» *AoB Plants,* vol. 12, n° %16, 2020.

[5] S. G. B. F. S. X. E. Y. W. Y.-X. C. Y.-F. &. X. Q.-L. Wu, « A Leaf Recognition Algorithm for Plant Classification Using Probabilistic Neural Network,» *IEEE International Symposium on Signal Processing and,* 2007.

[6] L.-D. Q. e. C.-N. N. Nghia Duong-Trung, «Learning Deep Transferability for Several Agricultural Classification Problems,» *International Journal of Advanced Computer Science and Applications(ijacsa),* vol. 10, n° %11, 2019.

[7] T. JASSMANN, «MOBILE LEAF CLASSIFICATION APPLICATION UTILIZING A CONVOLUTIONAL NEURAL NETWORK,» p. 71, 2015.

[8] A. B. e. J. W. Wei Di, Deep Learning Essentials, Packt, 2018, p. 271.

[9] M. N. R. K. G. D. e. K. T. Rikiya Yamashita, «Convolutional neural networks: an overview and application in radiology,» *Insights into Imaging ,* vol. 9, pp. 611-629, 2018.

[10] «Understanding Categorical Cross-Entropy Loss, Binary Cross-Entropy Loss, Softmax Loss, Logistic Loss, Focal Loss and all those confusing names,» 2018. [En ligne]. Available: https://gombru.github.io/2018/05/23/cross_entropy_loss/. [Accès le 05 2022].

[11] G. H. A. K. I. S. a. R. S. Nitish Srivastava, «Dropout: A Simple Way to Prevent Neural Networks from Overfitting,» *Journal of Machine Learning Research 15,* pp. 1929-1958, 2014.

[12] Y. Lecun, L. Bottou, Y. Bengio et P. Haffner, «Gradient-based learning applied to document recognition,» *Proceedings of the IEEE,* vol. 86, n° %111, 1998.

[13] A. Krizhevsky, I. Sutskever et G. E. Hinton, «ImageNet classification with deep convolutional neural networks,» *Communications of the ACM,* vol. 60, n° %16, 2017.

[14] K. S. e. A. Zisserman, «VERY DEEP CONVOLUTIONAL NETWORKS FOR LARGE-SCALE IMAGE RECOGNITION,» 2015.

[15] K. He, X. Zhang, S. Ren et J. Sun, «Deep Residual Learning for Image Recognition,» chez *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* , Las Vegas, NV, USA, 2016.

[16] C. Szegedy, Wei Liu, Yangqing Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke et A. Rabinovich, «Going deeper with convolutions,» chez *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Boston, MA, USA, 2015.

[17] A. Ng, «Convolutional Neural Networks,» [En ligne]. Available: https://www.coursera.org/learn/convolutional-neural-networks/home/welcome. [Accès le 04 2022].

[18] M. Z. B. C. D. K. W. W. T. W. M. A. e. H. A. Andrew G. Howard, «MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications,» 2017.

[19] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov et L.-C. Chen, «MobileNetV2: Inverted Residuals and Linear Bottlenecks,» chez *IEEE/CVF Conference on Computer Vision and Pattern Recognition* , Salt Lake City, UT, USA, 2018.

[20] C. Lepelaars, «L'Évolution des Architectures Mobiles CNN,» [En ligne]. Available: https://wandb.ai/wandb_fc/french/reports/L-volution-des-Architectures-Mobiles-CNN--Vmlldzo0NzM5NTc. [Accès le 05 2022].

[21] P. Pawara, «Plant recognition, detection, and counting with deep learning,» 2021.

[22] "MobileNet, optimisation de la convolution pour les réseaux de neurones embarqués.," [Online]. Available: https://www.quantmetry.com/blog/mobilenet-optimisation-de-la-convolution-pour-les-reseaux-de-neurones-embarques/#:~:text=L%27architecture%20des%20MobileNets%20est,une%20image%20de%20profondeur%201024. [Accessed 05 2022].

[23] «Categorical crossentropy,» [En ligne]. Available: https://peltarion.com/knowledge-center/documentation/modeling-view/build-an-ai-model/loss-functions/categorical-crossentropy. [Accès le 05 2022].

[24] M. Laurence, " Convolutional Neural Networks in TensorFlow," [Online]. Available: https://www.coursera.org/learn/convolutional-neural-networks-tensorflow/home/welcome. [Accessed 04 2022].

[25] H. K. M. S. Z. e. V. Ratheil, «Système d'identification de certaines maladies des plantes de riz avec les réseaux de neurones convolutifs,» p. 81, 2019.

[26] D. C. K. e. I. M. A. S. Krisna Hany Indrani, "Android Based Application for Rhizome Medicinal Plant Recognition Using SqueezeNet," vol. 4, 2020.

[27] M. R. S. e. P. M. Jana Wäldchen, «Automated plant species identification—Trends and future directions,» *PLOS COMPUTATIONAL BIOLOGY,* 2018.

[28] J. Krause, «CNN-BASED PLANT SPECIES CATEGORIZATION USING NATURAL,» p. 2020.

[29] I. T. K. e. S. Viriri, «Deep Learning for Improved Plant Classification,» 2018.

[30] «CNN with Data Augmentation,» [En ligne]. Available: https://hljames.github.io/dog-breed-classification/CNN_with_augmentation.html#data-augmentation. [Accès le 06 2022].

[31] "Classement des images," TensorFlow, [Online]. Available: https://www.tensorflow.org/tutorials/images/classification. [Accessed 04 2022].

[32] «Entraîner des modèles Keras à grande échelle avec Azure Machine Learning,» Microsoft, [En ligne]. Available: https://docs.microsoft.com/fr-fr/azure/machine-learning/how-to-train-keras. [Accès le 04 2022].

Résumé

Les applications de reconnaissance de plantes constituent un guide et un outil technologique moderne très précieux qui permet de répondre à la fois à la curiosité des simples amateurs de jardinage, des plantes, et passionnés de la nature en proposant avec précision l'espèce et le nom d'une plante. Elle apporte aussi une assistance précieuse aux explorateurs et scientifiques spécialistes du domaine. Notre projet Nabet-ID est une application de reconnaissance de plante permettant de retrouver/découvrir le nom des plantes en arabe, français, anglais et en darija, en analysant des photos de plantes sélectionnées dans la galerie photos d'un téléphone mobile ou simplement en prenant directement une photo d'une plante sans avoir forcément besoin d'une connexion internet. Nabet-ID est réalisée avec l'architecture de réseaux de neurones MobileNet V2, en exploitant les techniques de l'apprentissage par transfert, du réglage fin et de l'augmentation de données. Nous avons atteint sur un dataset de 10 classes un taux de 99% d'exactitude (accuracy) sur l'ensemble de test et 98% sur les ensembles de validation et de test. L'interface mobile de l'application Nabet-ID est développée en java sous android studio.

Mots clés : Identification, Classification, Reconnaissance, Plantes, Application Mobile, Réseau de Neurones Convolutifs, MobileNetV2, Apprentissage par Transfert, Augmentation de Données, Réglage Fin.

Abstract

Plant recognition applications constitute a guide and a very precious modern technological tool which makes it possible to respond to the curiosity of simple gardening amateurs, plants, and nature enthusiasts by proposing with precision the species and the name. of a plant. It also provides valuable assistance to explorers and scientists specializing in the field. Our Nabet-ID project is a plant recognition application allowing to find/discover the name of plants in Arabic, French, English and Darija, by analyzing photos of plants selected in the photo gallery of a mobile phone or simply by directly taking a picture of a plant without necessarily needing an internet connection. Nabet-ID is made with the MobileNet V2 neural network architecture, using the techniques of transfer learning, fine tuning and data augmentation. We achieved on a dataset of 10 classes a rate of 99% accuracy on the test set and 98% on the validation and test sets. The mobile interface of the Nabet-ID application is developed in java under android studio.

Keywords : Identification, Classification, Recognition, Plants, Mobile Application, Convolutional Neural Network, MobileNetV2, Transfer Learning, Data Augmentation, Fine Tuning.

ملخص

تشكل تطبيقات التعرف على النباتات دليلاً و أداة تكنولوجية حديثة قيّمة للغاية تجعل من الممكن الاستجابة لفضول هواة البستنة والنباتات وعشاق الطبيعة من خلال اقتراح نوع واسم النبات بدقة. كما تقدم مساعدة قيمة للمستكشفين والعلماء المتخصصين في هذا المجال. مشروعنا Nabet-ID هو تطبيق للتعرف على النباتات يسمح باكتشاف و التعرف على أسماء النباتات باللغة العربية والفرنسية والإنجليزية والدارجة ، من خلال تحليل صور النباتات المختارة في معرض الصور للهاتف المحمول أو عن طريق التقاط صورة مباشرة دون الحاجة بالضرورة إلى اتصال بالإنترنت. تم تصميم Nabet-ID باستخدام بنية الشبكة العصبية MobileNet V2 ، باستخدام تقنيات نقل التعلم والضبط الدقيق وزيادة البيانات. لقد حققنا في مجموعة بيانات من 10 فئات معدل دقة 99٪ (accuracy) في مجموعة الاختبار و 98٪ في مجموعات التحقق والاختبار. تم تطوير واجهة الهاتف المحمول لتطبيق Nabet-ID في Java تحت Android Studio.

الكلمات الرئيسية: تحديد الهوية ، التصنيف ، التعرف ، النباتات ، تطبيقات الهاتف المحمول ، الشبكة العصبية التلافيفية ، MobileNetV2 ، نقل التعلم ، زيادة البيانات ، الضبط الدقيق.