الجمهورية الجزائرية الديمقراطية الشعبية
République Algérienne Démocratique et Populaire
وزارة التعليم العالي والبحث العلمي
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique
جامعة غرداية
Université de Ghardaïa
كلية العلوم و التكنولوجيا
Faculté des Sciences et de la Technologie
قسم الرياضيات والاعلام الآلي
Département des Mathématiques et d'Informatique
مخبر الرياضيات و العلوم التطبيقية
Laboratoire des Mathématiques et Sciences Appliquées

*N° d'enregistrement*
/...../...../...../...../.....

# Mémoire

**Pour l'obtention du diplôme de master**

**Domaine:** Mathématiques et Informatique
**Filière:** Informatique
**Spécialité:** Systèmes Intelligents pour l'Extraction des Connaissances (SIEC)

**Présenté par : Nadir CHEIKH BABA & Hocine HOUDJEDJE**

## Thème

# Détection des Malwares Android à l'aide du Deep Learning

**Examiné par le jury composé de:**

| | | | |
|---|---|---|---|
| ***M.*** **Slimane BELLAOUAR** | **MCA** | **Univ.Ghardaïa** | **Examinateur** |
| ***M.*** **Youcef MAHDJOUB** | **MAA** | **Univ.Ghardaïa** | **Examinateur** |
| ***M.*** **Slimane OULADNAOUI** | **MCB** | **Univ.Ghardaïa** | **Encadrant** |

**Année universitaire 2020/2021**

# Reconnaissance


Tout d'abord, nous tenons à remercier "Allah" de nous avoir donné le courage et la volonté de mener à bien ce projet. Une sincère reconnaissance pour nous avoir éclairé.


Nous tenons à remercier le superviseur de cette thèse, *M.*Slimane OULADNAOUI, pour sa suggestion de ce sujet et ses précieux conseils, ainsi que pour ses efforts pour nous enseigner les années précédentes avec sa merveilleuse façon, nous vous apprécions et vous respectons grandement, merci beaucoup.

Nous tenons à remercier également les membres de jury :

*M.*Slimane BELLAOUAR et *M.*Youcef MAHDJOUB pour avoir accepté d'examiner ce travail.

Nous tenons également à exprimer nos sincères remerciements aux professeurs qui nous ont enseigné et inspiré dans notre parcours universitaire, et pour lesquels nous avons beaucoup d'appréciation et de gratitude pour leur éducation, leurs conseils et leurs encouragements, merci.

# Dédicace

On dit souvent que le trajet est aussi important que la destination. Les années d'études m'ont permis de bien comprendre la signification de cette phrase toute simple.
Ce parcours, en effet, ne s'est pas réalisé sans défis que j'ai relevés à travers l'effort et travail.
Je tiens à remercier ALLAH le tout-puissant de m'avoir donné la foi et de m'avoir permis d'en arriver là.
je dédie cette thèse:

**A mes chers parents.**

Je dédie ce modeste travail à mon cher père. Je te demande, ô Dieu, de lui pardonner, aie pitié de lui, et faite-le habiter dans les plus hauts rangs de votre paradis, Je le dédie à mon soutien dans cette vie, ma chère mère.

**A mes soeurs et frères.**

Qui ont été une source constante de soutien et d'encouragement, merci.

**À ma chère épouse.**

**A mes amis et à tous ceux qui croient en moi.**

**Hocine**

# Dédicace

Je tiens à remercier ALLAH le tout-puissant de m'avoir donné la foi et de m'avoir permis d'en arriver là.

Je dédie ce Mémoire:

**A mes chères parents.**

pour l'amour qu'ils m'ont toujours donné, leurs encouragements et toute l'aide qu'ils m'ont apportée durant mes études

**A ma chère Épouse et mes chères enfants.**

Avec tous mes sentiments de respect, d'amour, de gratitude et de reconnaissance pour tous les sacrifices déployés pour m' élever dignement et assurer mon éducation dans les meilleurs conditions

**A mes soeurs et frères.**

qui n'ont cessé d'être pour moi des exemples de persévérance, de courage et de générosité

**A mes amis et à tous ceux qui croient en moi.**

**Nadir**

## ملخص

على مدار السنوات القليلة الماضية ، أصبح نظام أندرويد أكثر أنظمة تشغيل الأجهزة المحمولة شهرة في العالم ، مما يجعله هدفًا رئيسيًا لمجرمي الإنترنت. لذلك يعد أمان هذه المنصة مسألة مهمة للغاية. على الرغم من كل ما يقدمه هذا النظام كآليات قوية لحماية منصته ، إلا أنه يعاني من العديد من العيوب ونقاط الضعف.

تهدف هذه الأطروحة إلى تنفيذ نموذج قائم على التعلم العميق قادر على الكشف تلقائيًا عما إذا كان تطبيق أندرويد مصابًا ببرامج ضارة أم لا دون الحاجة إلى تثبيته مسبقًا. نحن نجرب شبكة عصبية عميقة على مجموعة بيانات التعلم CCCS-CIC-AndMal-2020 ، والتي تتضمن 200000 عينة من التطبيقات الحميدة و 200000 من التطبيقات الخبيثة لما مجموعه 400000 تطبيق أندرويد. نتائج اختبار النموذج التي تم الحصول عليها مشجعة للغاية ، بدقة تصل إلى 98٪ خلال 25 حقبة فقط.



**الكلمات المفتاحية:** التعلم العميق ، البرامج الضارة ، أندرويد ، قاعدة بيانات CCCS-CIC-AndMal-2020.

## Abstract

Over the past few years, Android has become the world's most popular mobile operating system, making it a major target for cybercriminals. Therefore, the security of this platform is an extremely important issue. Despite all what this system offers as robust mechanisms to protect its platform, it still suffers from several flaws and vulnerabilities.

The present work aims to implement a model based on deep learning that can automatically detect whether an Android application is infected with malware or not without requiring its prior installation. We experiment with a deep neural network using the CCCS-CIC-AndMal-2020 learning dataset, which includes 200,000 samples of benign applications and 200,000 malignant applications for a total of 400,000 Android applications. The model testing results obtained are very encouraging, with an accuracy of up to 98% through 25 epochs only.



**Keywords:** Deep Learning, Malware , Android, CCCS-CIC-AndMal-2020 Dataset.

## Résumé

Au cours de ces dernières années, Android est devenu le système d'exploitation mobile le plus populaire au monde entier, et de ce fait une cible majeure pour les cybercriminels. Par conséquent, la sécurité de cette plateforme est une question extrêmement importante. En dépit de tout ce qu'offre ce système comme mécanismes robustes pour protéger sa plateforme, il souffre néanmoins de plusieurs failles et vulnérabilités.

Ce mémoire vise à mettre en œuvre un modèle basé sur le deep learning capable de détecter automatiquement si une application Android est infectée par un malware ou non sans exiger son installation préalable. Nous expérimentons un réseau de neuronnes profond sur l'ensemble de données d'apprentissage CCCS-CIC-AndMal-2020, qui comprend 200 000 échantillons d'applications bénignes et 200 000 applications malignes pour un total de 400 000 applications Android. Les résultats de teste du modèle obtenus sont très encourageants, avec une précision allant jusqu'à 98% a travers 25 époques seulement.



**Mots clés :** Apprentissage profond, malware, Android, Base de données CCCS-CIC-AndMal-2020.

# TABLE DES MATIÈRES

# TABLE DES FIGURES

# LISTE DES TABLEAUX

# INTRODUCTION

L'information aujourd'hui est devenue l'un des actifs les plus précieux, et par conséquent, elle est devenue gravement menacée par l'évolution des logiciels malveillants. Chaque année on remarque une augmentation exponentielle du nombre de logiciels malveillants, allant de 99,71 M logiciels malveillants détectés en 2012 jusqu'à 1282,22 M détectés en 2021 selon L'institut AV-TEST, qui recense chaque jour plus de 450 000 nouveaux malwares [l'Institute AVTest, 2021]. Le combat entre les créateurs des logiciels malveillants et les défenseurs pourrait ne jamais prendre fin. Au passé, les logiciels malveillants étaient destinés au divertissement. Mais il s'agit maintenant d'une industrie de malwares à but lucratif, qui utilise des nouvelles techniques améliorées d'obscurcissement complexes et de camouflage.

Les auteurs de logiciels malveillants essaient toujours d'écrire des programmes qui ne peuvent pas être facilement détectés. Donc les techniques de détection classiques basées sur les signatures des virus ne suffisent pas pour détecter des logiciels malveillants avancés. Généralement, lorsqu'un utilisateur télécharge une application dans son appareil mobile, il ne peut pas détecter si elle est bénigne ou maligne, tant que ne l'exécute pas sur son système, et qu'elle peut facilement l'endommager, s'il s'agit d'une application maligne.

Les techniques modernes basées sur le machine learning tel que les arbres de décision et les forêts aléatoires, aideront à détecter si une application est bénigne ou maligne avant de l'installer sur le système. Elles peuvent détecter les logiciels malveillants modernes et obscurcis avec grande précision. Prenons comme exemple, l'application qu'elle a été crée par Athiq Reheman Mohammed et al [Mohammed et al., 2019] qui a atteint une précision du modèle de 98,9% pour l'arbre de décision et 99.4% pour les forêts aléatoires. Toutefois, le problème majeur dans les algorithmes de machine learning, c'est l'obligation d'intégrer les connaissances des experts dans l'extraction des caractéristiques pour réduire la complexité des données et rendre les modèles d'apprentissages plus claires. C'est un processus généralement difficile et coûteux en termes de temps et d'expertise.

À l'opposé, les algorithmes de Deep Learning tentent d'apprendre des fonctionnalités de haut niveau à partir des données d'apprentissages sans aucune intervention. Ils ont montré des résultats prometteurs dans le domaine de détection des logiciels malveillants en utilisant différents types de réseau de neurones. Prenons comme exemple, le modèle proposé par Zhenlong Yuan

et al [Yuan et al., 2014] basé sur le Deep Belief Network (DBN), le modèle a atteint un niveau de précision élevé de plus de 96%.

Ce mémoire s'intéresse à créer et à évaluer un modèle de deep learning multicouches (DNN) pour la détection des malwares dans la plateforme des appareils mobiles la plus populaire Android en utilisent le nouvel ensemble de données de CCCS-CIC-AndMal-2020. Ce dernier comprend 200 000 échantillons d'applications Android bénignes et 200 000 applications Android malignes réparties en 14 catégories de logiciels malveillants (adware, ransomware, trojan-banker, etc.) et 191 familles de malware et en utilisant l'environnement Google Colab.

Pour réaliser cet objectif, nous avons délibérément organisé notre mémoire en trois principaux chapitres :

- Le chapitre 01 : contient des préliminaires, répartie en 04 sections, la première concerne le machine learning, la deuxième le deep learning ainsi que la définition de diffèrent algorithmes tel que ANN, CNN et RNN. La troisième est consacrée à l'architecture du système Android et sa sécurité. La dernière est sur les différents types de malware Android.
- Le chapitre 02 : contient l'état de l'art des techniques d'analyse des malwares (statique, dynamique, hybride), et des techniques de détection basées sur le machine learning et sur le deep learning, et les différents outils de détections de malware et une présentation de différents datasets utilisés dans ce domaine.
- Le chapitre 03 : ce chapitre contient notre propre expérience de créer un modèle de deep learning multicouche (DNN) pour la détection des malwares Android, avec une définition de son architecture, l'environnement de travail, la dernière partie est consacré pour l'expérimentation et l'interprétation des résultats obtenus.

# CHAPITRE 1

# PRÉLIMINAIRES

## 1.1 Introduction

Dans ce premier chapitre en va voir les définitions de quelque termes informatiques beaucoup utilisés récemment tels que : le Machine learning (l'apprentissage automatique), le Deep learning (l'apprentissage profond), le fameux système Android que nous utilisons comme plateforme dans la majorité de nos Smartphones ainsi que la sécurité du système Android, et à la fin les définitions de quelque Malwares les plus répandus qui menacent nos appareils mobiles.

## 1.2 Machine Learning

Depuis l'aube de l'ère technologique, les chercheurs rêvent de faire apprendre à la machine, comme le font les humains, comment raisonner et prendre des décisions intelligentes en tirer des généralisations et distiller des concepts à partir d'un ensemble d'information complexe sans avoir des instructions explicites. Le Machine learning fait référence à un aspect de cet objectif.

### 1.2.1 Définition

Le Machine Learning (l'apprentissage automatique) est une sous-catégorie de l'intelligence artificielle, c'est la science de la programmation des ordinateurs afin qu'ils puissent apprendre des données. Une définition plus générale : « L'apprentissage automatique est le domaine d'études qui donne aux ordinateurs la capacité d'apprendre sans être explicitement programmé ». —Arthur Samuel, 1959. Une autre définition plus répandue : « On dit qu'un programme informatique apprend de l'expérience E par rapport à une tâche T et une mesure de performance P, si sa performance sur T, telle que mesurée par P, s'améliore avec l'expérience E ». —Tom Mitchell, [Mitchell, 1997].

Traditionnellement, un programme effectue une tâche en suivant des instructions précises, alors qu'un système Machine Learning apprend d'une manière autonome à effectuer une tâche

ou à réaliser des prédictions à partir de données. Ses performances s'améliorent au fil de son entraînement à condition que l'algorithme soit exposé à suffisamment de données. [Géron, 2019]

### 1.2.2 Processus du Machine Learning

Pour développer un modèle de Machine Learning il faut passer par quatre étapes principales :

1. **Sélectionner et préparer l'ensemble de données d'entraînement** : Ces données seront utilisées pour alimenter le modèle de Machine Learning pour apprendre à résoudre un problème spécifique. Les données peuvent être étiquetées, afin d'indiquer au modèle les caractéristiques qu'il devra identifier. Elles peuvent aussi être non étiquetées, et le modèle devra repérer et extraire les caractéristiques récurrentes de lui-même. Ces données doivent être soigneusement préparées, organisées et nettoyées sinon l'entraînement du modèle de Machine Learning risque d'être biaisé.
2. **Sélectionner un algorithme à exécuter sur l'ensemble de données** d'entraînement. Le type d'algorithme à utiliser dépend du type et du volume de données d'entraînement et du type de problème à résoudre.
3. **Entraînement de l'algorithme**. Il s'agit d'un processus itératif, ou des paramètres sont utilisés à travers l'algorithme, et les résultats sont comparés avec ceux qu'il aurait dû produire. Les poids et le biais peuvent ensuite être ajustés pour améliorer la précision des résultats.
4. **Utilisation et amélioration du modèle**. On utilise le modèle sur de nouvelles données qui proviennent du problème à résoudre. Exemple : un modèle de Machine Learning conçu pour détecter des spams sera utilisé sur des emails [Data_Scientest, 2020].

### 1.2.3 Principales Tâches

La première étape pour déterminer quelle méthode de Machine Learning à utiliser consiste à spécifier ce que l'on va rechercher. D'après Hernán et ses collègues, il existe trois tâches principales de recherche en science des données : la description, la prédiction et l'inférence causale [Hernán et al., 2019].

#### Description

Lorsque l'intérêt est purement descriptif, nous avons besoin de très peu de théorie sur laquelle fonder notre recherche. Par exemple, nous pouvons être intéressés par la question de savoir si l'utilisation de punitions sévères par les parents est positivement associée à des problèmes de comportement chez les enfants. La description typique est que nous ne faisons aucune affirmation ou suggestion sur l'origine de cette association, ou comment cela pourrait être utilisé pour filtrer, sélectionner ou identifier des individus.

#### Prédiction

Le deuxième objectif de recherche est de prédire une variable particulière à partir d'une ou plusieurs autres variables. La prédiction nous permet d'identifier, de sélectionner des individus,

par exemple des adolescents qui risquent de devenir dépressifs ou des élèves du primaire qui bénéficieront de matériels d'apprentissage plus stimulants

#### Inférence Causale

Si le but de l'exploration est l'explication, nous avons besoin d'une étude sur les facteurs pouvant servir de causes et sur les variables qui sont les résultats ou les effets du processus causal. Sur cette base, nous pouvons développer des hypothèses par exemple : l'utilisation excessive des médias sociaux expose les adolescents au risque de se sentir seuls ; ou un sentiment accru de solitude amène un adolescent à passer plus de temps sur les réseaux sociaux. Il est essentiel ici que le changement de la variable X entraînera un changement de celle de Y [Jiang et al., 2020].

### 1.2.4 Types d'algorithmes du Machine Learning

Généralement, les algorithmes en machine learning peuvent être répartis en trois catégories : l'apprentissage supervisé, l'apprentissage non supervisés et l'apprentissage par renforcement.

#### Apprentissage supervisé

L'apprentissage supervisé est la technique la plus utilisée de Machine Learning. Au cours de la phase d'apprentissage, le système est alimenté par des ensembles de données étiquetés, qui indiquent au système quelle valeur de sortie est liée à quelle valeur d'entrée spécifique. Le modèle entraîné est ensuite présenté à des données de test qui ont été étiquetées, mais les étiquettes n'ont pas été révélées à l'algorithme d'apprentissage. Voir Figure 1.1 [Ayodele, 2010]

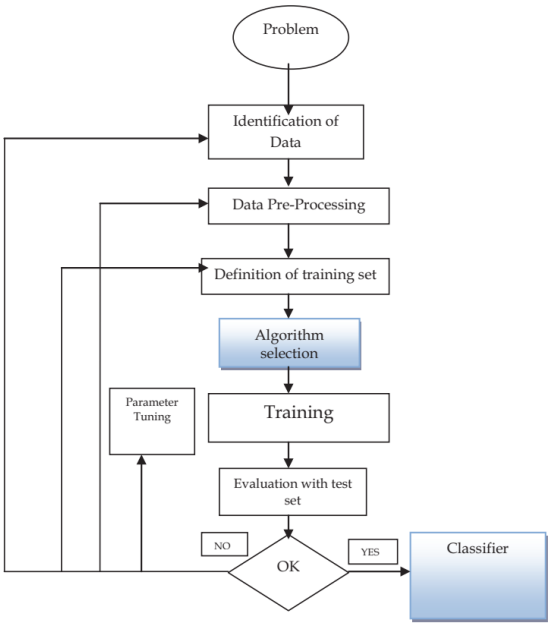


FIGURE 1.1 – Le processus d'apprentissage supervisé [Ayodele, 2010]

Il existe deux principaux types de techniques d'apprentissage supervisé. Le premier type est la

*classification*. Cette dernière prédise des sorties catégorielles, telles que si un certain objet est un chat ou un chien, si un mail est un spam ou non spam. Le deuxième type est la *régression*. Les techniques de régression prédisent des valeurs réelle, telle qu'un poids ou un volume... Voir Figure 1.2

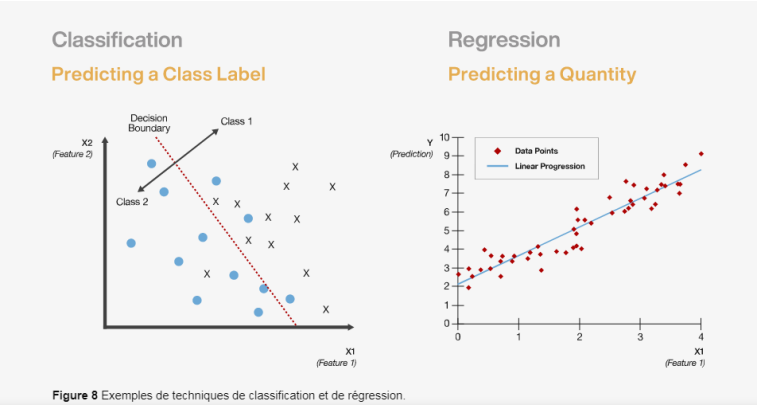


FIGURE 1.2 – Techniques de classification et de régression [C3.AI, 2021]

Voici Quelques algorithmes d'apprentissage supervisé les plus importants :

- La Régression linéaire (Linear Regression) ou logistique (Logistic Regression)
- k- plus proches voisins (k-Nearest Neighbors)
- Machine à vecteurs de support (Support Vector Machines (SVMs))
- Les Arbres de décision (Decision Trees )
- Les Forêts aléatoires (Random Forests)
- Classifieur bayésien naïf (Naive Bayes algorithm)
- Les réseaux de neurones (Neural networks) (Ils seront plus détaillés ci-dessous)

### Apprentissage non supervisé

L'apprentissage non supervisé est une technique de Machine Learning qui permet au système de travailler tout seul pour la détection de modèles et des descripteurs. Cependant, il n'y a pas de catégories ou d'étiquettes de sortie, l'algorithme essaye seul de modéliser les relations.

Ces algorithmes emploient des techniques sur les données d'entrée pour rechercher des règles, détecter des modèles, résumer et regrouper les points de données qui aident à tirer des informations significatives et à mieux décrire les données à l'utilisateur. voir Figure 1.3

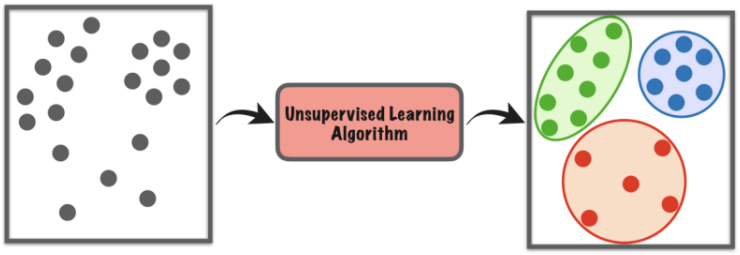


FIGURE 1.3 – Exemple de fonctionnement de l'apprentissage non supervisé [Jeffares, 2018]

Les algorithmes d'apprentissage non supervisé peuvent être regroupés en deux catégories principales : *le clustering* (regroupement) et *l'association*.

- **Clustering :** est une méthode de regroupement des objets de telle sorte que les objets présentant le plus de similitudes sont assemblés dans un groupe et présentent moins ou pas de similitudes avec les objets d'un autre groupe.
- **Association** Une règle d'association est une technique de trouver les relations entre les variables dans une grande base de données. Il détermine l'ensemble des éléments qui se produisent ensemble dans l'ensemble de données. La règle d'association rend la stratégie de marketing plus efficace. C'est-à-dire les personnes qui achètent un article X ont également tendance à acheter un article Y [Géron, 2019].

Voici quelques algorithmes d'apprentissage non supervisé les plus importants :

- K-moyennes (k-Means)
- Analyse en Composantes Principales (ACP) (Principal Component Analysis)
- Regroupement hiérarchique (Hierarchical Cluster Analysis) (HCA)
- Espérance-maximisation (Expectation Maximization)
- Noyau ACP (Kernel PCA)
- Locally-Linear Embedding (LLE)
- t-distributed Stochastic Neighbor Embedding (t-SNE)

### Apprentissage semi-supervisé

Il se situe entre l'apprentissage supervisé et non supervisé. C'est une autre technique de Machine Learning dans laquelle on dispose d'une grande quantité de données d'entrée et dont seules certaines données sont étiquetées. Voir Figure 1.4

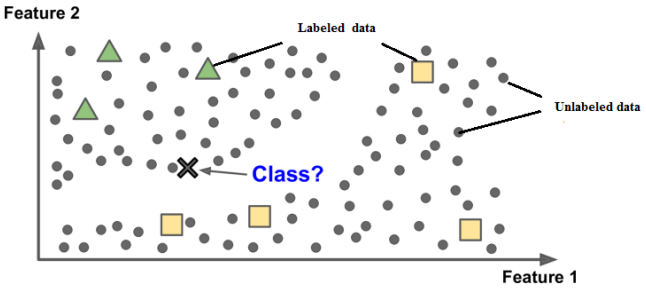


FIGURE 1.4 – Exemple de fonctionnement de l'apprentissage semi-supervisé [Machine_Learning_Mastery, 2021]

En effet, l'étiquetage des données peut être coûteux ou fastidieux et il nécessite parfois l'aide des experts du domaine. Alors que les données non étiquetées sont plus disponible et plus faciles à collecter et à stocker. Nous utilisons les techniques d'apprentissage supervisé pour découvrir et apprendre la structure des variables d'entrée. Nous utilisons les techniques d'apprentissage non supervisé pour faire des meilleures prédictions pour les données non étiquetées, après en va réinjecter ces données dans l'algorithme d'apprentissage supervisé en tant que données d'entrée. Le modèle sera ensuite exploité pour faire des prédictions sur de nouvelles données. [Machine_Learning_Mastery, 2021]

### Apprentissage par renforcement

Cette technique d'apprentissage consiste à ce que l'algorithme d'apprentissage (appelé agent) vise à utiliser les observations recueillies à partir de l'interaction avec son environnement de manière itérative et prendre des décisions (actions) pour maximiser la récompense ou minimise le risque (pénalités). Voir Figure 1.6

Dans le processus, l'agent apprend de ses expériences de l'environnement jusqu'à ce qu'il explore toute la gamme des états possibles et avoir la meilleure stratégie pour obtenir le plus de récompense au fil du temps. Une stratégie définit l'action que l'agent doit choisir lorsqu'il se trouve dans une situation donnée [Géron, 2019].

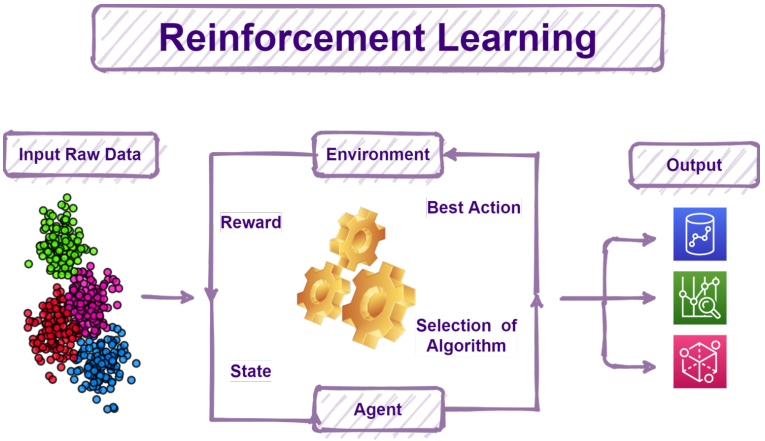


FIGURE 1.5 – Exemple de fonctionnement de l'apprentissage par renforcement [Cheng, 2020]

Voici quelques algorithmes d'apprentissage par renforcement les plus connus :

- Q-Learning
- Temporal Difference
- Learning (TD)
- Deep Adversarial Networks

### 1.2.5 Métriques pour évaluer l'algorithme de machine learning (Matrice de confusion)

L'évaluation de l'algorithme de mahine learning est une partie essentielle de tout projet. notre modèle peut nous donner des résultats satisfaisants lorsqu'il est évalué à l'aide d'une métrique. Il existe différents types de métriques d'évaluation notamment :

**La matrice de confusion** : Comme son nom l'indique, elle nous donne une matrice en sortie et décrit les performances complètes du modèle de prédiction [Mishra, 2018]. Il y a 4 termes importants dans une matrice de confusion :

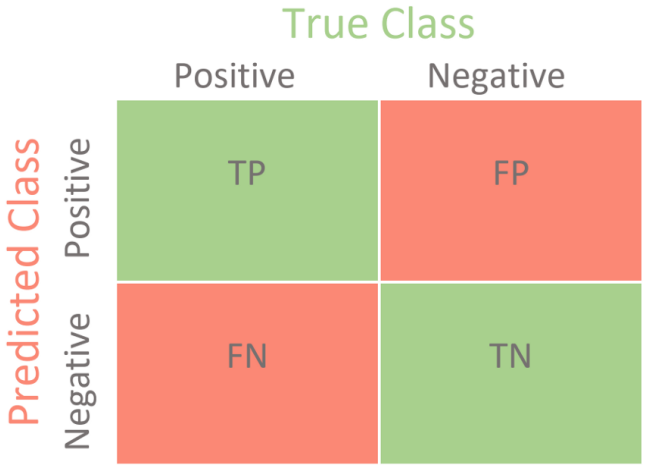


FIGURE 1.6 – Matrice de confusion [Mohajon, 2020]

- **True Positive** Les cas dans lesquels nous avons prédit **oui** et la sortie réelle était également **oui**. True positive rate est défini comme **TP**/ **(FN+TP)**.
- **True Negative** Les cas dans lesquels nous avons prédit **non** et la sortie réelle était **non**. True Negative Rate est défini comme **TN** /**(FP+TN)**.
- **False Positive** Les cas dans lesquels nous avons prédit **oui** et la sortie réelle était **non**. False Positive Rate est défini comme **FP** / **(FP+TN)**.
- **False Negative** Les cas dans lesquels nous avons prédit **non** et la sortie réelle était **oui**. False Negative Rate est défini comme **FN** / **(FN+TP)**.

## 1.3 Deep Learning

Le Deep Learning (Apprentissage profond) est un sous-domaine du machine Learning qui représente les algorithmes inspirés de la structure et du fonctionnement du cerveau humain, L'unité fondamentale du cerveau est le neurone. Voir Figure 1.7

Un grain de riz du cerveau humain contient plus de 10 000 neurones, dont chacun forme en moyenne 6 000 connexions avec d'autres neurones qui apprennent en ajustant les poids de leurs liens. Ce gigantesque réseau biologique nous permet de découvrir le monde qui nous entoure [Buduma and Locascio, 2017].

La création des réseaux de neurones artificiels a été motivée par cette observation. Mais, comment des réseaux de ce type général peuvent-ils apprendre les représentations internes compliquées qui sont requises pour effectuer des tâches difficiles telles que la reconnaissance des objets et la compréhension du langage ? Le deep Learning cherche à répondre à cette question.

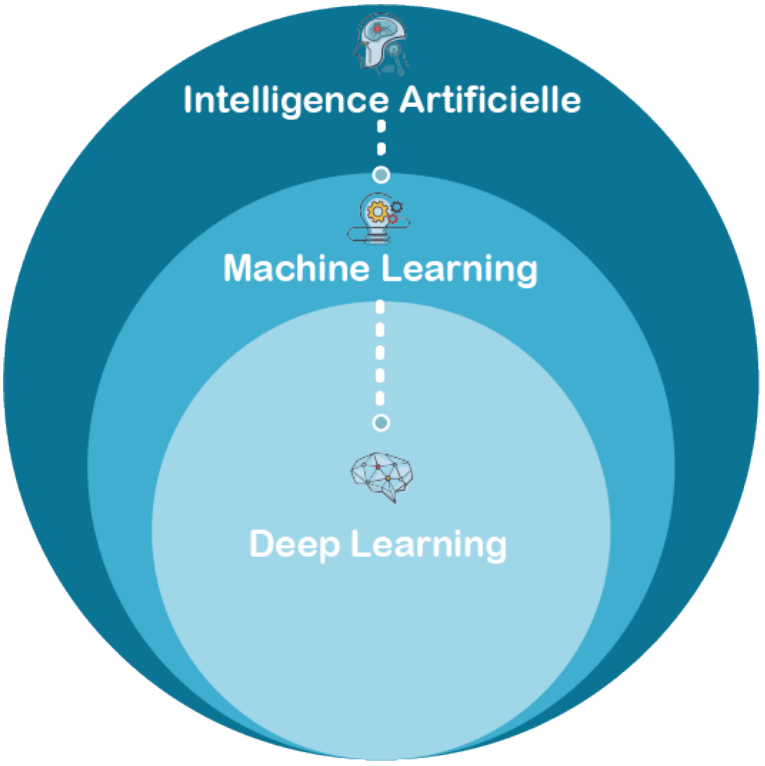


FIGURE 1.7 – Image représentant la relation entre le Machine Learning et le Deep Learning [Data_Scientist_Bial-R, 2020]

On peut résumer la différence entre le Machine Learning et le Deep Learning dans les quelques points importants ci-dessous :

#### Dépendances des données

La différence la plus importante entre le Deep Learning et le Machine Learning traditionnel réside dans ses performances à mesure que l'échelle des données augmente. Lorsque les données

sont petites, les algorithmes de Machine Learning excellent. Les algorithmes de Deep Learning ont besoin d'une grande quantité de données pour apprendre parfaitement.

### Dépendances matérielles

Contrairement aux algorithmes d'apprentissage automatique traditionnels, qui peuvent fonctionner sur des machines bas de gamme, ceux du deep learning dépendent fortement de la puissance des machines. Ils exigent des outils de calcul puissants pour leur fonctionnement à cause de grande quantité d'opérations de multiplication matricielle qu'ils effectuent. Ces opérations peuvent être optimisées efficacement à l'aide d'un GPU (unité de traitement graphique).

### Structure des fonctionnalités

En ML traditionnel, la sélection de caractéristiques est un processus consistant à intégrer les connaissances des experts du domaine dans le choix des caractéristiques pour réduire la complexité des données et rendre les modèles plus claires pour le bon fonctionnement des algorithmes. Ce processus est difficile et coûteux en termes de temps et d'expertise. Les algorithmes de Deep Learning tentent d'apprendre des fonctionnalités de haut niveau à partir des données.

### Approche de résolution de problèmes

En machine learning, il est généralement recommandé de décomposer un problème en différentes parties, et de les résoudre individuellement et de les combiner pour obtenir le résultat. Le Deep Learning en revanche préconise de résoudre le problème de bout en bout c'est à dire le modèle apprend toutes les étapes entre la phase d'entrée initiale et le résultat de sortie final simultanément.

### Temps d'exécution

Un algorithme de Deep Learning prend habituellement énormément de temps pour s'entraîner en raison qu'il y a beaucoup de paramètres à fixer. Son exécution prend plus de temps que d'habitude allant des fois jusqu'à plusieurs jours. Alors qu'un algorithme machine learning prend comparativement beaucoup moins de temps à s'entraîner, allant de quelques secondes à quelques heures.

### Interprétabilité

L'interprétabilité des modèles de Deep Learning a toujours été un facteur limitant pour les cas d'utilisation nécessitant des explications sur les fonctionnalités impliquées dans les modèles et c'est le cas pour de nombreuses industries telles que les services financiers.

Les institutions financières préfèrent les modèles structurels faciles à interpréter par les humains, c'est pourquoi les modèles de Deep Learning au sein de ces industries ont été adoptés lentement, ils préfèrent généralement les méthodes statistiques classiques telles que les modèles linéaires, les modèles bayésiens et les modèles de machine learning traditionnels tels que les arbres de décision qui sont facilement explicables et interprétés. Les modèles de Deep Learning peuvent atteindre une précision élevée, mais au détriment d'une abstraction élevée [Shaikh, 2017].

Les Réseaux de neurones se présentent sous plusieurs formes différentes, dans notre mémoire on va voir les trois réseaux de neurones les plus populaires, notamment les Réseaux de neurones artificiels et profond, les réseaux de neurones de convolution et les Réseaux de neurones récurrents, et chacun présente des avantages pour des cas d'utilisation spécifiques.

### 1.3.1 Réseaux de neurones artificiels

Le réseau de neurone artificiel (ANN : Artificial Neural Network) constitue la base du deep Learning. C'est une technologie qui simule le réseau de neurones biologiques.

Chaque neurone reçoit des signaux électrochimiques d'autres neurones au niveau de ses dendrites. Si ces entrées électriques sont suffisamment puissantes pour activer le neurone, alors le neurone activé se déclenché et transmet le signal à travers de son axone aux dendrites d'autres neurones. Ces neurones connectés peuvent également se déclencher, poursuivant ainsi le processus de transmission du message. Voir Figure 1.8

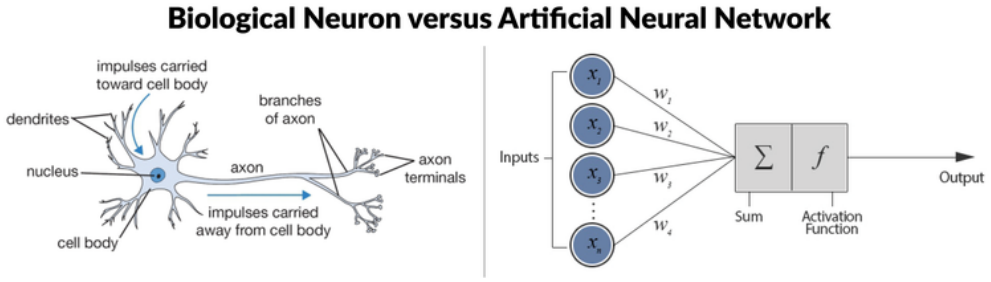


FIGURE 1.8 – Image représentant l'anatomie d'un neurone biologique et un neurone artificiel [Brian Mwandau, 2020]

Par exemple un réseau de neurones de base (un perceptron), qui contient au moins deux couches, une couche d'entrée (input layer) et une couche de sortie (output layer) et qui effectue une simple sommation pondérée des entrées. Les valeurs $x_1 ; x_2 ; \ldots ; x_n$ dans la figure 7 sont les entrées de notre réseau de neurones, que nous pouvons les considérer comme des vecteurs de caractéristiques d'entrée. Ces entrées par exemple pourraient être des vecteurs utilisés pour quantifier le contenu d'une image de manière systématique (les histogrammes de couleurs, les motifs, etc.). Chaque entrée $x$ est connectée à un neurone via un vecteur de poids $W$ composé de $w_1 ; w_2 ; \ldots ; w_n$, ce qui signifie que pour chaque entrée $x_i$ nous avons un poids associé $w_i$. le nœud de sortie prend la somme pondérée des $x_i$ (les entrées) et des $w_i$ (les poids) somme $\sum_{i=1}^{n} w_i x_i$, est lui applique une fonction d'activation $f$ qui va déterminer si le neurone se déclenche ou non [Rosebrock, 2017].

#### Fonctions d'activation

Une fonction d'activation est une fonction mathématique appliquée à un signal en sortie d'un neurone artificiel. Ce terme de fonction d'activation vient du terme biologique potentiel d'activation, une fois le seuil de stimulation est atteint, il entraîne une réponse du neurone. La

fonction d'activation est souvent une fonction non linéaire. Parmi les fonctions les plus utilisés et qu'on va utiliser quelque une dans la partie implémentation on peut citer :

1. **La fonction Heaviside** : Est la fonction d'activation la plus simple, utilisée par l'algorithme de Perceptron (algorithme d'apprentissage supervisé inventé en 1957 par Frank Rosenblatt), il s'agit d'une fonction de seuil qui revoit 1 comme résultat si la somme pondérée est positive et 0 sinon. Voir Figure 1.9

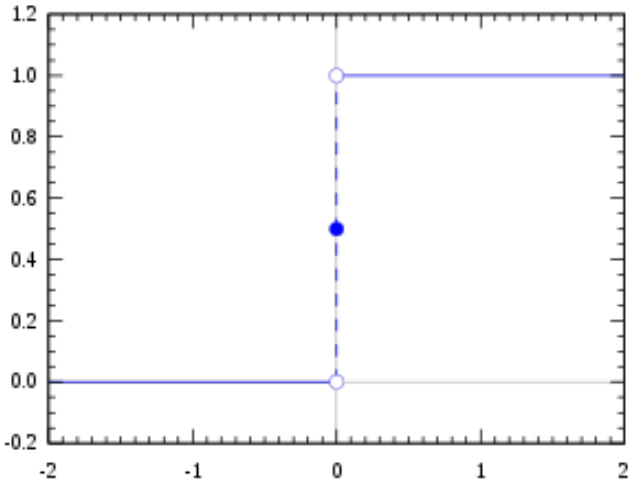


FIGURE 1.9 – La fonction Heaviside qui ressemble à une marche d'escalier [Gabormelli, 2021]

   Cependant, malgré qu'elle est simple et facile à utiliser, mais elle est moins utilisée en deep Learning.

2. **La fonction sigmoïde** (la fonction logistique) : Souvent utilisés dans les réseaux de neurones. elle est utilisé pour la classification binaire, elle est définie comme :

$$s(t) = 1/(1 + e^{-t})$$

   où $e$ est la constante exponentielle. Nous fixons d'abord la variable $t$ à la somme pondérée des entrées du neurone, puis la transmettons à la fonction sigmoïde qui fournit des valeurs comprises entre 0 et 1. Voir Figure 1.10

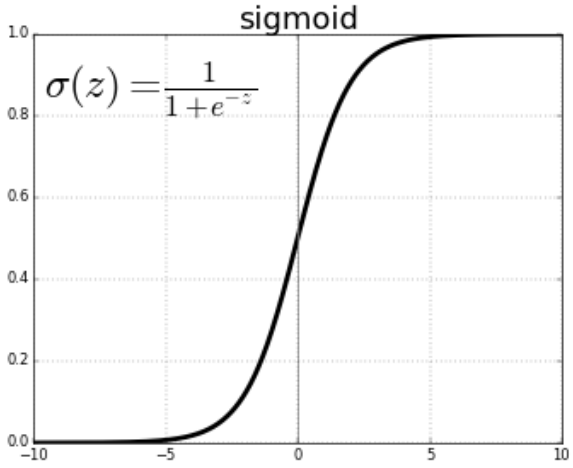


FIGURE 1.10 – la fonction sigmoïde [Ml4a, 2021]

3. **La fonction ReLU (l'unité linéaire rectifiée)** : Est la fonction d'activation la plus simple et la plus utilisée elle est définie par :

$$R(z) = max(0, z)$$

Cette fonction laisse toutes les valeurs positives passer inchangées et annule les valeurs négatives. La plupart des réseaux de neurones actuels utilisent ReLU dans leurs couches cachées ou l'une de ses variantes proches. Voir Figure 1.11

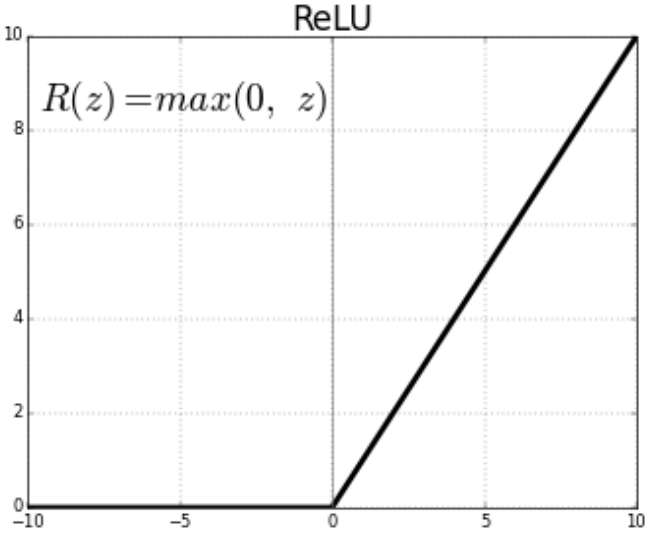


FIGURE 1.11 – la fonction ReLU [Ml4a, 2021]

4. **La fonction Softmax** : Aussi appelé fonction exponentielle normalisée, est une généralisation de la fonction logistique, elle transforme un vecteur de $K$ valeurs réelles en un vecteur de probabilités dont la somme est égale à 1. Elle est utilisée comme fonction d'activation dans la couche de sortie des modèles de réseaux de neurone pour les problèmes

de classification multiclasses afin de classer les entrées en plusieurs catégories. Voir Figure 1.12

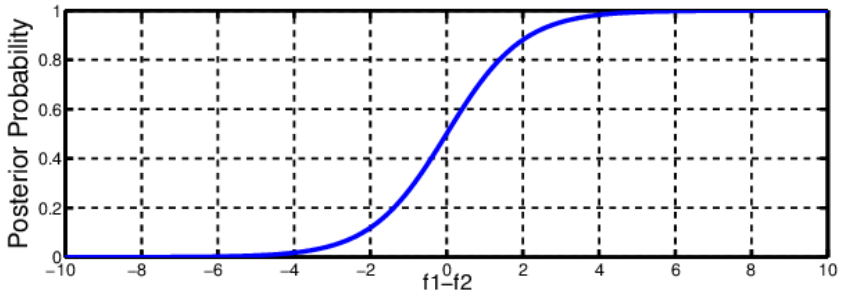


FIGURE 1.12 – la fonction Softmax [Chen et al., 2017]

Il existe encore d'autres fonctions d'activations très utiles comme la Tangente hyperbolique, Leaky ReLU, ELU, SoftPlus, etc

**Feedforward et Rétropropagation**

Les réseaux de neurones artificiels sont des réseaux Feedforward, ce qui signifie que les connexions entre les nœuds ne forment pas un cycle ; c'est-à-dire les informations se déplacent dans le réseau dans un seul sens vers l'avant, depuis la couche d'entrée et à travers les couches cachés, et vers la couche de sortie.

La *rétropropagation* est une technique très importante dans l'entraînement d'un réseau de neurone. C'est le réglage fin des poids des neurones en fonction du taux d'erreur obtenu par apport à l'époque (itération) précédente de l'entraînement (l'entraînement se fait à plusieurs époques), les poids dans la première époque sont définis aléatoirement.

La technique la plus populaire dans La rétro-propagation est la *descente de gradient* (gradient descent). Son principe est que chacun des poids du réseau de neurone reçoit une mise à jour proportionnelle à la dérivée partielle de la fonction d'erreur par rapport au poids actuel à chaque itération de l'apprentissage.

On peut considérer un gradient comme la pente d'une fonction. Plus le gradient est élevé, plus la pente est raide et plus un modèle peut apprendre rapidement. Mais si le gradient est nul, la pente s'annule et le modèle va s'arrêter d'apprendre. Un bon réglage des poids permet de réduire les taux d'erreur et rend le modèle plus fiable et plus précis.

### 1.3.2 Réseau de neurones de convolution

Les réseaux de neurones de convolution (CNN) sont inspirés de l'organisation fonctionnelle du cortex visuel ; ils ont été introduits pour la première fois dans les débuts des années 1980 par le chercheur Yann LeCun qui introduit la première version des CNN, appelée LeNet. Ce dernier est un simple réseau de neurones de convolution, qui a été appliqué avec succès pour identifier les numéros de codes postals manuscrits.

Les réseaux de neurones de convolution, n'ont pas pu être étendu davantage en raison de manque de puissance de calcul et de manque d'ensemble de données d'entraînement jusqu'à

l'année 2012, suite à l'invention d'AlexNet (réseaux de neurones multicouches) par Alex Krizhevsky et ces collègues est la disponibilité de grands ensembles de données tel que ImageNet (Base de données contient des millions d'images étiquetées) et une abondance de ressources informatiques utilisant des GPU, tous ces avantages ont permis aux chercheurs de relancer les CNN.

Les CNN désignent donc une sous-catégorie de réseaux de neurones profonds très adaptés aux modèles de classification d'images et réputés en être les plus performants. Le rôle principal des CNN est de réduire les images sous une forme plus facile à traiter, sans perdre les attributs essentielles pour obtenir une bonne prédiction.

### Principe de fonctionnement

- **Couche d'entrée** (Input layer) : L'utilisateur fournit en entrée un ensemble d'images, chaque image est sous la forme d'une matrice de pixels, Celle-ci dispose de 3 dimensions (Deux dimensions pour une image en niveaux de gris, Une troisième dimension, de profondeur 3 pour représenter les couleurs fondamentales RVB). Voir Figure 1.13

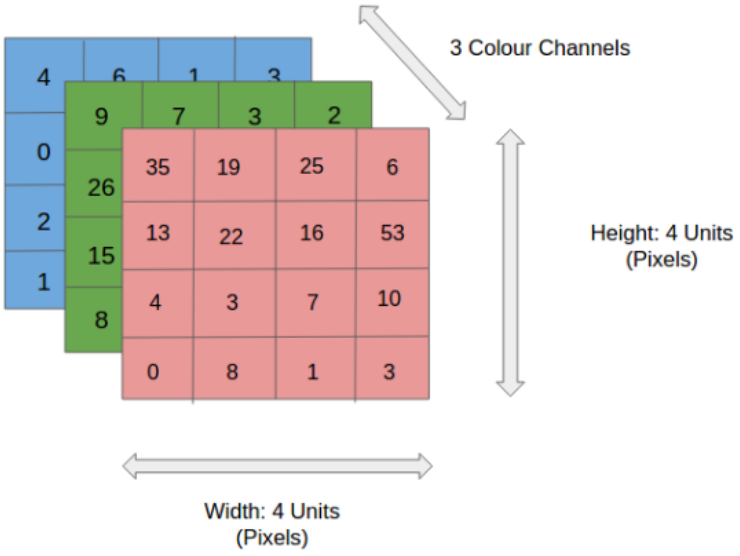


FIGURE 1.13 – Exemple d'image de 3 dimensions [Saha, 2018]

Pour plus de simplicité, restons-en aux images en niveaux de gris. Juste Après cette couche en a la couche de convolution. Voir Figure 1.14

- **Couche de convolution** On définit la taille de la fenêtre de filtre de convolution appeler aussi noyau de convolution, dans l'exemple la fenêtre de filtre est une matrice 3×3, qui se déplace progressivement de la gauche vers la droite d'une case jusqu'à arriver au bout de l'image.

  À chaque portion del'image rencontrée, un calcul de convolution s'effectue permettant d'obtenir en sortie une carte d'attributs (feature map) La première couche de convolution extrait généralement les caractéristiques de base telles que les bords horizontaux ou diagonaux. Cette sortie est transmise à la couche suivante qui détecte les caractéristiques

plus complexes telles que les coins ou les bords combinatoires, qu'à la fin le réseau peut identifier des caractéristiques encore plus complexes telles que des objets, des visages, ... Chaque couche utilise plusieurs fonctions d'activation qui génèrent une valeur d'activation qui sera transmise à la couche suivante. La fonction d'activation la plus utilisée dans les CNN [Buduma and Locascio, 2017].

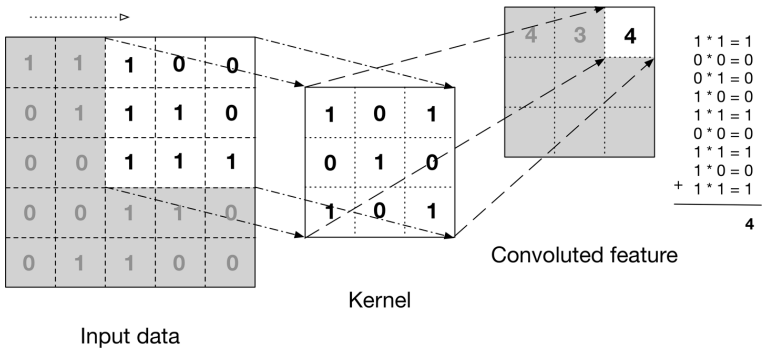


FIGURE 1.14 – Exemple de fonctionnement de la convolution [Giuseppe Ciaburro, 2021]

- **Couche de Max/Average pooling :** Le Max/Average pooling est une opération généralement appliquée entre deux couches de convolution. Celle-ci reçoit en entrée les cartes d'attributs formées en sortie de la couche de convolution et son rôle est de réduire la taille des images, tout en préservant leurs attributs les plus essentielles. Il s'agit d'une réduction des dimensions pour minimiser le coût de traitement.

  Le pooling ressemble à convolution, la seule différence est que pour chacune des régions balayées par le filtre, le pooling prendra le maximum/moyenne créant ainsi une nouvelle matrice de sortie où chaque élément de la matrice correspondra aux maximums/moyenne de chaque région balayée. Il fonctionne également comme un suppresseur de bruit. Voir Figure 1.15

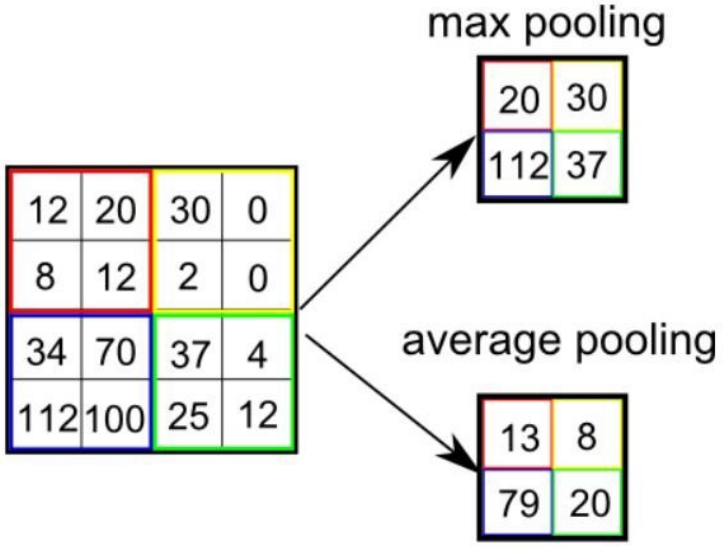


FIGURE 1.15 – Exemple de fonctionnement de Max pooling [Saha, 2018]

- **Couche entièrement connectée (Fully Connected) :** Les couches entièrement connectées forment les dernières couches du réseau. L'entrée de la couche Fully Connected présente la sortie de la couche finale de pooling ou de convolution, qui est aplatie (convertir les valeurs de la matrice tridimensionnelle de sortie en un vecteur d'une seule dimension), ce vecteur aplati est ensuite connecté aux couches entièrement connectées qui sont identiques aux réseaux de neurones artificiels et effectuent les mêmes opérations mathématiques.

  Dans la dernière couche de sortie on utilise la fonction d'activation Softmax pour classer les images dans plusieurs classes [Saha, 2018]. Voir Figure 1.16

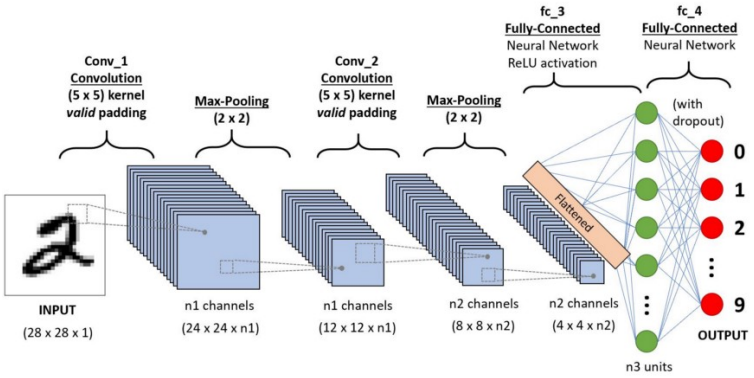


FIGURE 1.16 – Les séquences de CNN pour classer les chiffres manuscrits [Saha, 2018]

### 1.3.3 Réseau de neurones récurrent

Les êtres humains ne commencent pas chaque fois leur réflexion à partir de zéro, en lisant par exemple un article, nous comprenons chaque mot en fonction de notre compréhension des mots précédents. Les RNN ont la capacité de se souvenir des données entrantes, ce qui leur permet d'être très précis pour les prochaines prédictions. Pour cette raison, ils sont préférés pour le traitement des données séquentielles ou de séries chronologiques telles que la traduction, la reconnaissance vocale, le traitement vidéo, le sous-titrage d'images.

À l'opposé des réseaux Feedforward, dans le réseau récurrent les neurones ont des connexions récurrentes, qui propagent les informations entre les neurones de la même couche. La couche récurrente est connectée à un flux d'informations de chaque neurone vers les autres neurones de sa couche y compris lui-même. Voir Figure 1.17

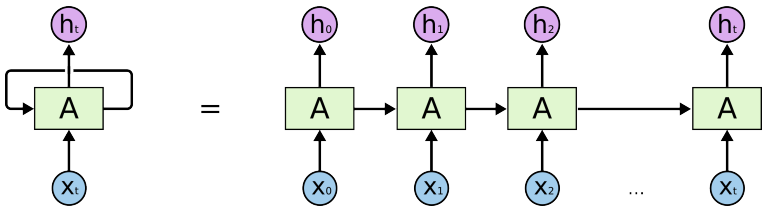


FIGURE 1.17 – Architecture d'un réseau de neurone récurrent [Ma, 2016]

En observant la figure ci-dessus. On voit sur la partie gauche que cette couche prend en entrée une donnée chronologique $x_t$($t$ est l'indice de temps de l'observation dans la séquence) et retourne un vecteur de sortie $h_t$. On remarque qu'il existe une boucle de récurrence. La partie droite de la figure, nous présente le fonctionnement de traitement de toute la séquence d'observations de $x_0$ jusqu'à $x_t$. On peut alors remarquer que pour prédire la sortie $h_1$, la couche récurrente va non seulement observer le contenu de $x_1$ (comme le cas des réseaux feed-forward), mais également obtenir de l'information venue des états passés allant de la boîte A du temps $t = 0$ à la boite A du temps $t = 1$. En résumé, le neurone d'un RNN encode la connaissance acquise grâce aux observations passées et la prenne en compte en plus de l'observation courante pour fournir une information de sortie qui soit représentative du contenu global de la séquence [Donges, 2018]. Mais il existe quelques problèmes qui se posent au cours de l'entrainement d'un RNN de base qui sont :

- Le premier problème est quand on utilise une fonction d'activation telle que tanh (Tangente hyperbolique, sort des valeurs entre -1 et 1) dans les couches cachées, Voir Figure 1.18 . Soit le calcul du gradient en rétro-propagation dépasse les limites de calcul et provoque des mises à jour des poids très importants qui s'accumulent a travers l'entrainement du modèle et produit en fin une explosion du gradient (exploding gradient). Soit Au contraire pendant l'entrainement le gradient s'affaibli et tendre vers 0 suite à la mise à jour des valeurs très petites des poids initiaux utilisés, qui produit une dissipation du gradient (vanishing gradient), ce dernier est le problème le plus majeur dans les RNN.
- Le deuxième problème dans les RNN standard, c'est qu'ils ont besoin d'une mémoire interne importante pour stocker l'ensemble des informations des grandes séquences. Prenant comme exemple la complétion automatique de la phrase suivante : « La vieillesse est si longue qu'il ne faut pas la commencer trop tôt », Le RNN n'arrive pas à prédire le mot tôt, car il n'a pas retenu le mot vieillesse. Pour arriver à déterminer le mot tôt, il faut que le RNN possède une mémoire plus large.

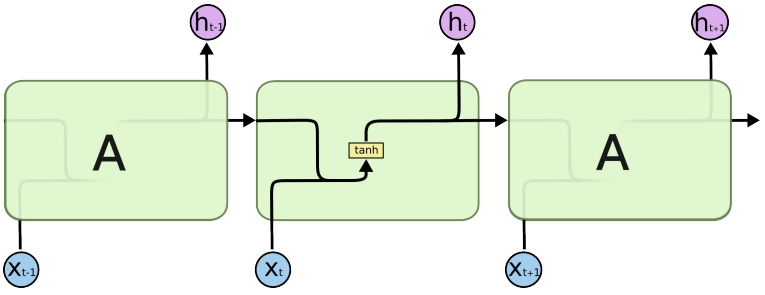


FIGURE 1.18 – Réseau de neurone récurrent standard avec fonction d'activation Tanh [Kirillov, 2018]

Pour résoudre ces problèmes, il existe plusieurs solutions qui ont favorisé les RNN pour être les plus populaires le domaine de traitement de séquences, parmi les solutions en trouvent : [Peters et al., 2018]

- Les LSTM (Long short-term memory).
- Les GRU (Gated Recurrent Unit).

- Les Bi-LSTM (Bi-directional long short term memory)
- LES MD-LSTM (Multi-dimensional LSTM)
- le modèle ELMo (Embeddings from Language Models)
- Les TCN (Temporel Convolutional Networks)

Dans cette partie, nous nous limitons à un seul modele les LSTM qui sont les plus populaires.

### LSTM (Long short-term memory)

Les LSTM sont une extension des RNN pour étendre essentiellement sa mémoire. Ils permettent aux RNN de mémoriser les entrées séquentielles à longue terme. Elles contiennent des informations dans une mémoire (qui ressemble à une mémoire d'ordinateur) ; Elles peuvent lire, écrire et supprimer des informations de la mémoire, cette mémoire peut être considérée comme une cellule fermée. En fonction de l'importance de l'information, la cellule décide de stocker ou de supprimer des informations, cette importance se fait par le biais des poids, qui sont attribué par l'algorithme à partir de l'entrainement.

Une cellule d'un LSTM est composée de trois portes et de deux états. [Yan, 2016] Les portes sont des zones de calculs qui régulent le flot d'informations en réalisant des actions très spécifiques et en exécutant des fonctions tel que la fonction logistique Sigmoïde et la fonction Tanh.

- **Forget gate (porte d'oubli)** : à la capacité de supprimer l'information, quand celle-ci n'est pas importante.
- **Input gate (porte d'entrée)** : à la capacité à prendre en compte de nouvelles informations utiles.
- **Output gate (porte de sortie)** : pour laisser affecter la sortie à un instant $t$ sachant l'information de la forget et l'input gate.
- **Hidden state (état caché)** : contient des informations sur les entrées précédentes du réseau et sert aux prédictions
- **Cell state (état de la cellule)** : pour enregistrer l'information que le LSTM a jugé pertinente.

Ci-dessous une image qui illustre le fonctionnement d'un LSTM. [Yan, 2016]

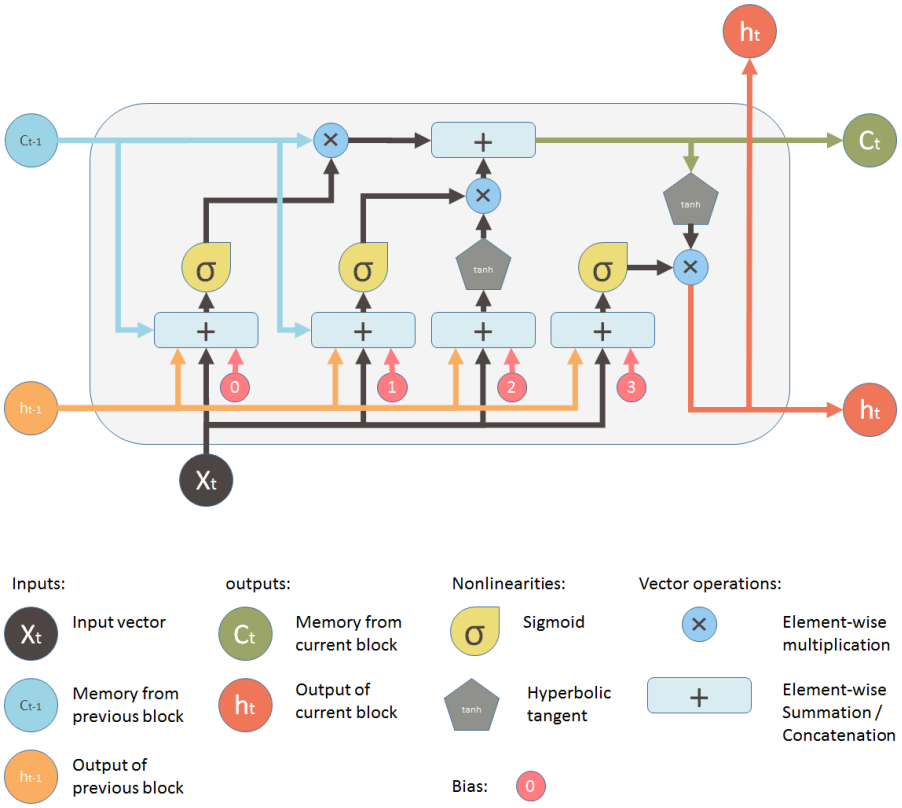


FIGURE 1.19 – Architecture d'un LSTM [Yan, 2016]

### Les types de RNN

Il existe 05 types de réseaux de neurone récurrents. Selon notre cas d'utilisation, nous pouvant configurer notre RNN pour gérer les entrées et les sorties différemment. La principale raison pour laquelle les réseaux récurrents sont plus excitants est qu'ils nous permettent d'opérer sur des séquences de vecteurs, une séquence en entrée et en sortie, ou les deux ensembles dans le cas général. L'image si dessous représente les différentes architectures des RNN : Chaque rectangle représente un vecteur, Les vecteurs rouge sont des entrée, les vecteurs verts contiennent l'état du RNN, les vecteurs bleu sont des sortie.

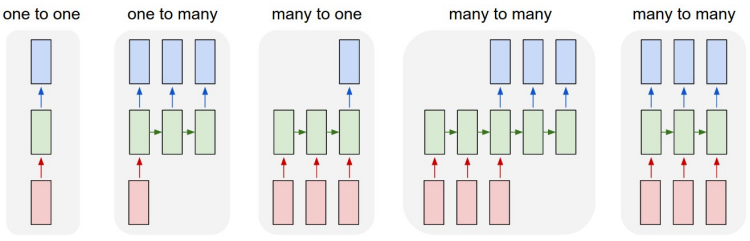


FIGURE 1.20 – Différentes architectures des RNN [Karpathy, 2015]

- **One to One** : est le type de réseau de neurones le plus basique et traditionnel donnant une seule sortie pour une seule entrée. (ex : classification d'images).
- **One-to-Many** : Traite une taille fixe d'informations en entrée qui donne une séquence

de données en sortie. (ex : sous-titrage d'images, prend l'image en entrée et génère une phrase de mots)

- **Many-to-One** : Prend une séquence d'informations en entrée et génère une taille fixe de la sortie. (ex : analyse des sentiments).
- **Many-to-Many** : Prend une séquence d'informations en entrée et traite les sorties récurrentes comme une séquence de données. (ex : traduction automatique).
- **Many-to-Many (synchronisés)** : Entrée et sortie de séquence synchronisée. (ex : classification vidéo pour étiqueter chaque image de la vidéo)

Il existe d'autre types de réseau de neurone, Nous en citons ici quelques-uns :

- **Le Transformer** : est un nouveau modèle de deep learning utilisé principalement dans le domaine du traitement automatique des langues.
- **L'auto-encoder** : est un réseau de neurones artificiels utilisé pour l'apprentissage non supervisé, qui permettent de construire une nouvelle représentation d'un jeu de données.
- **Le Generative adversarial network (GAN)** : est un réseau de neurones artificiels utilisé pour l'apprentissage non supervisé qui permet de générer des images avec un fort degré de réalisme.

### 1.3.4 Généralisation, régularisation et optimisation

Pour avoir un modele de deep learning plus fiable, robuste et plus précis, il existe plusieurs techniques pour améliorer ses performances que nous développons ci-après.

#### La généralisation

La capacité d'un réseau à généraliser est l'aspect le plus important dans le deep learning. Il permet au modèle d'être réutilisé dans une variété de domaines. Deux phénomènes qui contribuent largement à une mauvaise généralisation sont le sur-apprentissage (overfitting) et le sou-sapprentissage (underfitting). Voir Figure 1.21.

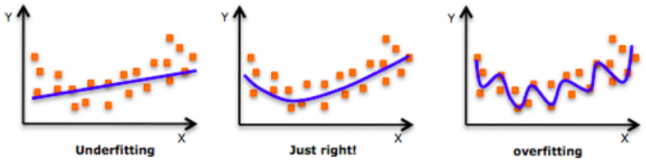


FIGURE 1.21 – Illustration des problèmes d'apprentissage [Datarobot, 2021]

1. **Le sur-apprentissage (Overfitting)** Très courant, ce problème se produit lorsqu'un modèle apprend trop bien l'ensemble de données d'entraînement, il apprend les détails et le bruit dans les données d'apprentissage en tant que concepts par le modèle, dans la mesure où cela a un impact négatif sur les performances du modèle sur de nouvelles données.

2. **Le sous-apprentissage (Underfitting)** à l'inverse le sous-apprentissage se produit lorsqu'un modèle échoue à apprendre suffisamment le problème et qui fonctionne mal sur l'ensemble de données d'apprentissage. Il ne peut donc généraliser sur de nouvelles données. Le sous-apprentissage est facile à détecter à l'aide d'une bonne métrique de performance.

Les solutions possibles au problème de généralisation consistent à obtenir plus de données et de réduire le bruit dans les données d'apprentissage et à simplifier l'algorithme utilisé en réduisant le nombre des paramètres et caractéristiques utilisés, ou de régulariser le modèle. Le succès du modèle d'entrainement dépend en grande partie de la qualité de la généralisation.

### La régularisation

La régularisation est une technique qui apporte des modifications à l'algorithme d'apprentissage de sorte que le modèle se généralise mieux et fonctionne bien non seulement sur les données d'entraînement, mais aussi sur les nouvelles entrées [Chollet, 2017]. Il existe de très nombreuses formes de régularisation. Nous citons parmi ces stratégies les techniques ci-dessous.

1. **Dropout** C'est l'une des techniques la plus fréquemment utilisée. À chaque itération, le dropout, sélectionne aléatoirement certains nœuds et les supprime avec toutes leurs connexions entrantes et sortantes. Par conséquent, chaque itération a un ensemble différent de nœuds et cela se traduit par un ensemble différent de sorties. Le dropout peut être appliquée à la fois à la couche d'entrée qu'aux couches cachées. Le but est d'éviter le sur-apprentissage en modifiant explicitement l'architecture du réseau au moment de l'apprentissage, ce qui permet de mieux généraliser. Comme le montre l'image ci-dessous

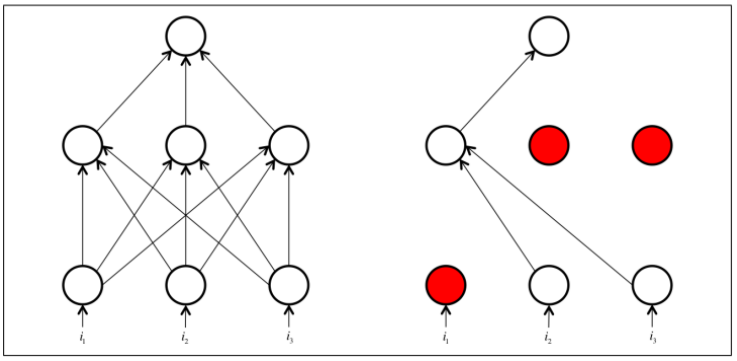

FIGURE 1.22 – Dropout [Buduma and Locascio, 2017]

2. **Data Augmentation** Une autre manière de réduire le sur-apprentissage et d'améliorer la généralisation est d'augmenter la taille des données d'apprentissage. Comme la disponibilité des données étiquetées est coûteuse, on fait recourir à une certaine transformation sur l'ensemble des données existantes. Par exemple effectuer les opérations : rotation de l'image, retournement, mise à l'échelle, décalage, etc. Voir Figure 1.23


FIGURE 1.23 – Data Augmentation [Srihari, 2021]

3. **Régularisation de poids par L1 & L2**. Pour simplifier un réseau de neurone, il faut imposer des contraintes sur la complexité du réseau en forçant ses poids à ne prendre que de petites valeurs, ce qui rend la distribution des valeurs de poids plus régulière. En ajoutant un coût à la fonction de perte (loss function), qui est une fonction qui évalue l'écart entre la valeur réelle à prédire et la valeur prédite par le réseau de neurones. Plus le résultat de cette fonction est minimisé, plus le réseau de neurones est performant. Le coût est associé surtout au poids importants.

    - *Régularisation L1 :* Le coût ajouté est proportionnel à la valeur absolue des coefficients de pondération.
    - *Régularisation L2 :* Le coût ajouté est proportionnel au carré de la valeur des coefficients de pondération.

### L'optimisation

L'optimisation maximise la productivité, la résistance, la fiabilité, l'efficacité et la performance du modèle et accélère l'entrainement d'une part ; et d'autre, elle minimise le facteur d'erreur. La première étape consiste à définir un modèle et une tâche. Le premier est composé d'une architecture et de paramètres qui déterminent la précision avec laquelle le modèle exécute la tâche. Le choix de la fonction d'optimisation est déterminant.

Les optimiseurs façonnent le modèle dans sa forme la plus précise possible en utilisant des paramètres tels que les poids et le taux d'apprentissage. Le plus populaire de cette famille est la *descente de gradient* (gradient descent), déjà évoqué. La descente de gradient est un algorithme d'optimisation basé sur une fonction convexe et ajuste ses paramètres de manière itérative pour minimiser la fonction de perte à des valeurs minimums.

Souvent, le problème rencontré lors de l'optimisation est le blocage sur des minima locaux [Kathuria, 2018]. Lorsqu'on traite des grandes quantités de données, il est possible d'atteindre une zone qui nous semble la plus basse possible pour la fonction de perte, mais il ne s'agit en réalité que d'un minimum local. Un bon taux d'apprentissage nous évite justement de rester coincés dans les minima locaux. Voir Figure 1.24

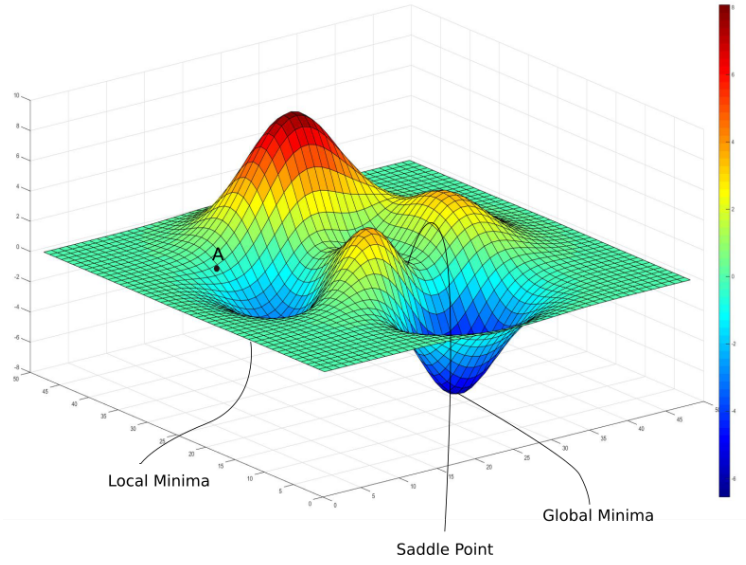


FIGURE 1.24 – Minimum local et minimum global [TechTalks, 2021]

**Le taux d'apprentissage** : Il détermine à quelle vitesse (pas) nous nous dirigeons vers les poids optimaux [Doshi, 2019]. 1.25

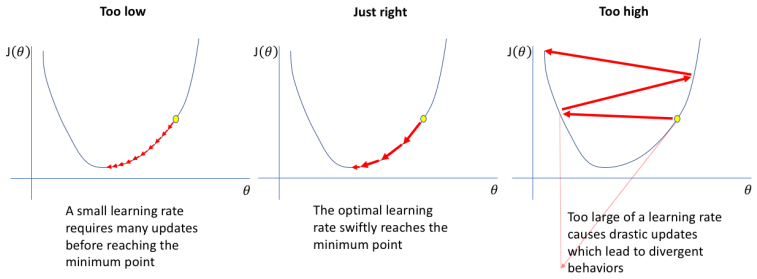


FIGURE 1.25 – illustration les différents taux d'apprentissage [Jordan, 2018]

**Le Batch size (taille du lot) :** Le batch size correspond au nombre de points de données utilisés pour entraîner un modèle à chaque itération. Son choix est important pour assurer la convergence vers un bon ensemble de paramètres, et pour la généralisation. Notons que de petits lots impliquent des mises à jour nombreuses et rapides. Par contre, les plus grandes, peut améliorer l'efficacité de parallélisme des GPU et peut accélérer l'apprentissage, mais pas assez volumineuses pour ne pas saturer la mémoire GPU. Il existe plusieurs optimiseurs qui sont plus rapide et plus performent que l'optimiseur de descente de gradient classique, les plus populaires sont : Stochastic Gradient Descent (SGD), Mini-Batch Gradient Descent, Adam, Optimisation Momentum, Nesterov Accelerated Gradient, AdaGrad, AdaDelta et RMSProp.

## 1.4 Système Android

Android est un système d'exploitation open source mobile dérivé d'une version adaptée du noyau Linux et d'autres composants logiciels. Il est conçu pour les appareils mobiles à écran tactile tels que les smartphones et les tablettes. La plupart des appareils mobiles qui utilisent Android sont livrés avec un logiciel propriétaire supplémentaire préinstallé, notamment Google Mobile Services (GMS). Ce dernier inclut des applications de base telles que Google Chrome, la plate-forme de distribution numérique Google Play, etc [Android, 2021].

### 1.4.1 Historique

Voici les points remarquables dans la chronologie d'Android.

- En 2003, Création par Andy Rubin est ses collègues de la société Android Inc. visant à développer un système d'exploitation pour les appareils photo numériques [Gargenta, 2011].
- En 2004, le projet a changé pour devenir un système d'exploitation pour smartphones.
- En 2005 Android Inc., a été racheté par la société américaine Google Inc. L'équipe chargée d'Android chez Google, a décidé de fonder son projet sur le système Linux.
- Le 5 novembre 2007, Google a annoncé la création de l'Open Handset Alliance, un consortium de dizaines d'entreprises de technologie et de téléphonie mobile, comme Intel Corporation, Motorola Inc., NVIDIA Corporation, LG Electronics, Samsung Électronique, etc. Ce consortium vise de développer et de promouvoir Android en tant que système d'exploitation open source gratuit avec prise en charge des nouvelles applications.
- En 2008, le kit de développement SDK Android 1.0 est publié. Le téléphone G1, fabriqué par HTC, vendu par l'opérateur sans fil T-Mobile USA.
- En 2009 large diffusion de la nouvelle version du système d'exploitation Android
- En 2010, les appareils mobiles d'Android sont les deuxièmes plus vendu derrière le Blackberry.
- En 2012, Android est devenu le système d'exploitation le plus populaire pour les appareils mobiles, dépassant l'iOS d'Apple.
- En 2020, environ 75% des appareils mobiles fonctionnent sous Android [Encyclopaedia_Britannica, 2020].

### 1.4.2 Architecture de la plateforme Android

Le système d'exploitation Android est composé de plusieurs couches. Chaque couche à ses propres caractéristiques et objectifs. Les couches ne sont pas nettement séparées mais s'infiltrent souvent les unes dans les autres [Android_Platform, 2021]. La Figure 1.26 nous donne une illustration complète de l'architecture de la plateforme Android

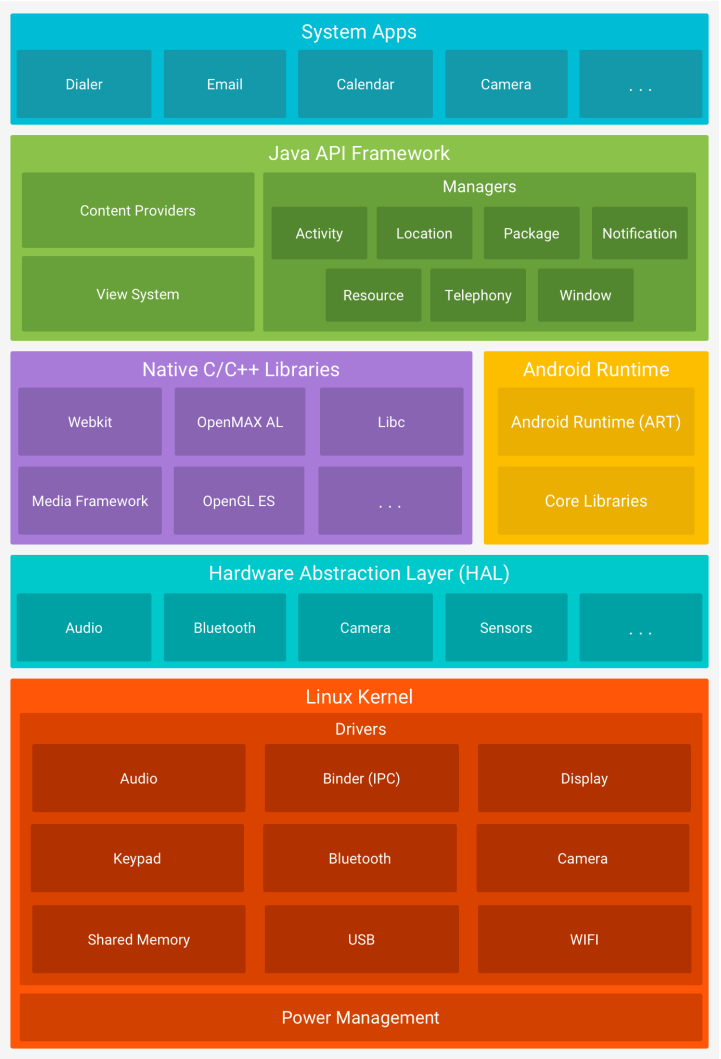


FIGURE 1.26 – Architecture de la plateforme Android [Android_Platform, 2021]

### Le noyau Linux

Le noyau Linux représente le cœur de la plateforme Android. L'utilisation d'un noyau Linux permet à Android de tirer parti des principales fonctionnalités de sécurité et de performances.

### Hardware Abstraction Layer

La couche d'abstraction matérielle (HAL) sépare la partie logique du matériel ; elle fournit des interfaces standards qui relient les périphériques matériels à la couche de Java API framework. Elle inclut plusieurs modules de bibliothèque, dont chacun implémente une interface pour un type spécifique de composant matériel.

### Android Runtime

ART (Android Runtime) est conçu pour exécuter plusieurs machines virtuelles sur des appareils à faible mémoire. Pour les appareils exécutant Android version 5.0 (API niveau 21) ou supérieur, chaque application s'exécute dans son propre processus et avec sa propre instance d'ART.

#### Native C/C++ Libraries

Cette couche contient plusieurs bibliothèques natives écrites en langage C et C++ qui sont indispensables pour Les composants et les services de base du système Android et pour le fonctionnement des applications. Par exemple la bibliothèque OpenGL ES est nécessaire pour ajouter aux applications la prise en charge du dessin et la manipulation graphiques en 2D et 3D.

#### Java API Framework

Java API Framework est une interface de programmation d'applications pour Android ; elle constitue un environnement très riche qui fournit de nombreuses fonctionnalités pour aider les développeurs a créé leurs propres applications Android. Parmi ces fonctionnalités :

- Un système de mise en page (layout) riche et extensible pour créer l'interface utilisateur d'une application, (les listes, les zones de texte, les boutons, etc)
- Un gestionnaire de ressources, qui donne un accès à des ressources et des fichiers supplémentaires qu'on utilise dans notre code, tels que les bitmaps, les définitions de disposition, les instructions d'animation, etc.
- Un gestionnaire de notifications qui permet aux applications d'afficher des alertes personnalisées dans la barre d'état.
- Un gestionnaire d'activités qui gère le cycle de vie des applications et fournit une pile de navigation commune pour les applications.
- Des fournisseurs de contenu qui aident les applications à gérer l'accès aux données stockées par elles-mêmes ou stockées par d'autres applications, et à partager des données avec d'autres applications.

#### Applications systèmes

Contient la totalité des applications Android qui sont, soit préinstallés sur l'appareil comme la messagerie électronique, le SMS, le calendrier, le navigateur web, les contacts, etc., soit téléchargés [Android_Platform, 2021].

### 1.4.3 Sécurité de système Android

Vu le développement de nos sociétés, il est devenu nécessaire d'avoir un appareil mobile, qui est une sorte de mini-ordinateur, et qui sert effectuer diverses tâches : faire des appels, envoyer des SMS, se connecter à internet, prendre des photos, opérations bancaires, etc. Nos appareils mobiles stockent des informations personnelles précieuses et des données sensibles telles que les identifiants, les photos, les e-mails, etc. La confidentialité de ces données est d'une importance primordiale pour l'utilisateur, car elles pourraient être utilisées à des fins d'imitation d'identité, d'harcèlement et d'escroquerie.

Au cours des dernières années, le système Android est devenu le système d'exploitation mobile le plus populaire du marché, qui par conséquent devenu une cible majeure pour les cybercriminels. Ce qui rend la sécurité de ces appareils est extrêmement importante.

Pour cela Android utilise des pratiques de sécurité de pointe et des mécanismes en collaboration avec les développeurs pour assurer la sécurité de sa la plate-forme, parmi ces mécanismes [Android_Safety, 2021, Android_Security, 2020] :

- L'utilisation du noyau Linux dans la plate-forme Android lui assure une certaine stabilité et sécurité et ce grâce à son historique de recherches, d'attaques et de corrections constantes par des milliers de développeurs.
- Un système d'exploitation multi-utilisateur qui assure l'Isolement des processus et la protection des ressources des utilisateurs les unes des autres (fichiers, ressources CPU. Mémoire,. . . ).
- Une fonction de communication interprocessus sécurisée (IPC) utilisant une technique appelée Sandbox, pour permettre une communication sécurisée entre les applications s'exécutant dans différents processus, et qui empêche l'application malveillante de nuire au système Android et à d'autres applications.
- Un modèle d'autorisations d'exécution basé sur l'utilisateur qui donne l'habilié aux utilisateurs de choisir les autorisations à accorder aux applications qui utilisent des API protégées sur l'appareil tel que l'appareil photo, le GPS, le Bluetooth, etc. Avant l'installation de toute application, l'utilisateur reçoit un message détaillé sur les différentes autorisations demandées par l'application.
- Une partition système en lecture seule qui contient le noyau d'Android ainsi que les libraries du système, le runtime de l'application, le framework d'application et les applications. Plus un mode sans échec sécurisé.
- Un démarrage vérifié à partir de la racine matérielle de confiance jusqu'à la partition système. Avant d'exécuter une étape, chaque étape vérifie de manière cryptée l'intégrité et l'authenticité de l'étape suivante.
- Android fournit aux applications un ensemble d'API cryptographiques qui content une base de codage standard et couramment utilisées telles que AES, RSA, DSA et SHA.
- Sur Android, les autorisations de la racine root sont accordées seulement au noyau et à un petit sous-ensemble d'applications principales, mais l'utilisateur ou l'application qui dispose d'une autorisation root a le droit de modifié le système d'exploitation, le noyau ou toute autre application y comprend les données ce qui augmente l'exposition de la sécurité de l'appareil aux applications malveillantes et aux failles des applications. Pour protégé les données d'application des utilisateurs root, les applications peuvent ajouter une couche de protection des données en utilisant le cryptage avec une clé stockée hors de l'appareil.
- Android fourni un cryptage complet du système de fichiers, utilisé un mot de passe pour l'appareil peut protéger la clé de chiffrement dans le cas d'un vol ou d'une perte de cette appareil, la modification du bootloader ou du système d'exploitation n'est pas suffisante pour accéder aux données d'utilisateur sans avoir le mot de passe utilisateur. En plus Android, fournit des fonctionnalités d'administration (API) aux appareils. Comme la messagerie intégrer dans les appareils, à partir des mots de passe alphanumériques ou des codes PIN numériques, les administrateurs peuvent également effacer à distance ou

restaurer les paramètres d'usine par défaut des appareils perdus ou volés.

- Demande d'une autorisation explicite aux applications tierces demandant l'utilisation d'API sensibles aux coûts qui génère un coût pour l'utilisateur ou le réseau lorsqu'elle est utilisée, tel que la Téléphonie, les SMS/MMS, les Réseaux de Données, etc.
- Restreindre l'accès de bas niveau à la carte SIM par les applications. toutes les informations personnelles et les communications avec la carte SIM sont géré par le système d'exploitation.
- Toutes les applications exécutées sur la plate-forme Android doivent être signées par les développeurs cela permet aux développeurs d'identifier l'auteur de l'application.
- Un espace sécurisé pour les développeurs pour atteindre facilement les utilisateurs Android et les clients, Google Play Protect fournit une vérification des licences d'application, une analyse de la sécurité des applications et bien d'autres services de sécurité.
- Une navigation internet sécurisée par Google avec une notification lors de l'ouverture d'un site Web ou un fichier douteux.
- Un programme de mis à jour périodique pour fournir des correctifs aux failles de sécurité.
- Un formulaire de rapport de vulnérabilité de sécurité à la disposition des développeurs et utilisateurs Android pour informer l'équipe de sécurité Android des problèmes courants liés à la sécurité

  L'objectif de tout cet arsenal de mécanismes de sécurité et techniques de protection offert par le système Android est pour assurer la sécurité de sa plate-forme contre les logiciels et les programmes malveillants (Malwares), mais Qu'est-ce qu'un malware ?

## 1.5 Mobile Malware

Le terme malware est l'abréviation du terme malicious software (en anglais), c'est un terme générique qui décrit tout programme ou code malveillant nuisible aux systèmes. Il signifie aussi tout type de logiciels indésirables destinés à nuire, envahir, endommager ou désactiver les systèmes informatiques. Les malwares mobiles désignent tous logiciels malveillants spécialement conçus pour cibler les appareils mobiles. Ils sont utilisés par les cybercriminels comme des armes pour atteindre leurs objectifs comme [Kremer et al., 2019] :

- L'espionnage et le vol d'informations confidentielles.
- Le harcèlement et l'envoi des courriers et messages indésirables.
- L'escroquerie et le détournement d'argent.
- La pénétration des réseaux publics et personnels.
- La paralysation des systèmes.

Ces attaques conduisent souvent à de graves dommages et à des pertes financières importantes.

### 1.5.1 Types des malwares mobiles

Les cybercriminels utilisent diverses tactiques pour infecter les appareils mobiles. Il est important de comprendre les différents types de menaces de logiciels malveillants mobiles. Voici quelques types les plus courants [Kaspersky_Mobile_Security, 2021].

#### Mobile Adware

C'est un script ou un programme installé sur l'appareil mobile souvent sans le consentement de l'utilisateur, et qui affiche des publicités (Pop-ups) indésirables sur le mobile. Les programmes publicitaires auront tendance à vous proposer des publicités contextuelles, peuvent modifier la page d'accueil du navigateur, ajouter d'autres spywares.

#### Mobile Spyware

Un logiciel chargé par l'utilisateur sur l'appareil mobile, qui incluent généralement un élément de programme espion, qui collecte discrètement des informations de la victime tel que, ses détails de localisation, ses contacts, son utilisation d'internet, ses identifiants de comptes de messagerie ou des sites de commerce électronique, et les envoie à un tiers.

#### Mobile Ransomware

Le ransomware est un type d'attaque de malware qui crypte les données importantes d'une victime telles que les documents, les photos et les vidéos, jusqu'à ce qu'un paiement soit effectué à l'attaquant (rançon), généralement en Bitcoin parce qu'il est intraçable. Une fois que la victime a payé la rançon, des codes d'accès lui sont fournis pour lui permettre de déverrouiller ses données, si le paiement de la rançon n'est pas effectué, l'attaquant généralement publie ces données privés sur des sites de fuite de données (DLS, data leak sites) ou les verrouiller à vie.

#### Banking Malware

Sont des codes intégrer dans des logiciels malveillants destiné à compromettre les utilisateurs qui préfèrent mener toutes leurs activités bancaires à partir de leurs appareils mobiles tel que les transferts d'argent et les paiements de factures, le e-commerce, etc. Les appareils des victimes sont infiltrer par le malware, qui collecte les informations de connexion et les mots de passe bancaires, et les renvois ensuite à un serveur de commande et de contrôle. Les Banking Malware sont en croissance très rapide, car c'est un moyen facile de gagner de l'argent.

#### MMS Malware

Certains pirates informatiques ont exploité une vulnérabilité dans la médiathèque d'Android (libstagefrigh), pour envoyer un message MMS contenant des logiciels malveillants à n'importe quel numéro de mobile, qui leur permet d'accéder à la racine Root des appareils mobile et d'avoir enfin un accès complet aux données sans aucune intervention de l'utilisateur.

#### SMS Trojans (chevaux de Troie)

Sont des programmes de chevaux de Troie crée par des cybercriminels, leurs buts est d'infecté les appareils mobiles des victimes en envoyant des SMS à des certains numéros de téléphone surtaxés à travers le monde, qui augmente leurs factures téléphoniques. Ou d'intercepter des messages texte des victimes contenant des informations financières, puis envoyer une copie du message texte par e-mail aux cybercriminels, en leur donnant toutes les informations qu'ils avaient besoin [Kaspersky_Mobile_Security, 2021].

#### Drive by Downloads

Est une forme dangereuse de malware mobile connu sous le nom de conduire à partir de téléchargement ; la source principale de ces malwares est l'ouverture de mauvais e-mail, de visiter un site Web malveillant ou à partir des réseaux sociaux. Après l'installation automatique de ces variantes sur votre appareil mobile, Ils peuvent déclencher une série de menaces, tel que des spywares, des malwares, des adwares, ou au pire des cas des Bots qui auras un accès complet à l'appareil et à son contenu, et commencent à communiquer et à recevoir des instructions d'un ou plusieurs serveurs de commande et de contrôle. Ils peuvent utiliser l'appareil pour effectuer des tâches très dangereuses telles que la propagation dans des réseaux d'organisations et puis l'envoi des virus à d'autres personnes [Kaspersky, 2021].

### 1.5.2 Principaux signes d'infection

Il existe quelques signes qui indiquent l'infection d'un appareil mobile, parmi ces signes [Malware_Bytes, 2021] :

- L'apparition soudaine de pop-ups avec des publicités invasives.
- L'augmentation anormale de l'utilisation des données mobile, à cause de la possibilité d'affichage de Pop-ups indésirables ou l'envoi des informations personnel par des logiciels malveillants.
- Des frais supplémentaire sur votre facture téléphonique.
- La batterie se décharge rapidement.
- L'envoi involontairement d'e-mails et des SMS étranges à vos contacts depuis votre Appareil mobile.
- L'augmentation de chaleur de l'appareil mobile, qui signifie généralement que de nombreuses activités gourmandes en ressources force le processeur a travaillé sans arrêt.
- L'existence d'applications qui non pas été téléchargées, après un téléchargement d'une application apparemment légitime et qui contienne des malwares. Ces malwares téléchargent automatiquement d'autres applications malveillantes.
- L'activation automatique des connexions Wi-Fi et Internet par les malwares.

### 1.5.3 Test de détection des logiciels espions pour Android

Le laboratoire de l'Institut de recherche indépendant pour la sécurité informatique de l'Allemagne AVTEST GmbH [AV_Test_Institute, 2021] a fait un test daté de juillet 2021, de détection de 29 sortes de logiciels espions y compris les stalkerwares (logiciel de surveillance, sort de spyware, utilisé pour le cyberharcèlement, largement utilisé par des gens pour espionner leurs conjoints). 18 applications de sécurité pour Android ont participé au test, tel que Bitdefender, ESET, Antiy, Kaspersky, etc., pour montrer leurs capacités à détecter les logiciels espions. Les applications Antiy, Bitdefender et Trend Micro ont pu détecter la totalité des logiciel espion, suivies de très près par ESET et Kaspersky, avec un taux de détection de 96.6%, suivies des autres applications de sécurité avec des taux de détection acceptables situant entre 93.1% et 58.6% Google Play Protect, l'application de sécurité interne d'Android, n'a pu détecter que 9 logiciel espion sur 29 avec un taux de détection de 31 pour cent seulement [AV_Test, 2021]. Voir la Figure 1.27


| Fabricant | Produit | Applications espionnes détectées | Applications espionnes non détectées | Taux de détection |
|---|---|---|---|---|
| Antiy | AVL | 29 | 0 | 100% |
| Bitdefender | Mobile Security | 29 | 0 | 100% |
| Trend Micro | Mobile Security | 29 | 0 | 100% |
| ESET | Mobile Security | 28 | 1 | 96,6% |
| Kaspersky | Internet Security for Android | 28 | 1 | 96,6% |
| F-Secure | SAFE | 27 | 2 | 93,1% |
| G DATA | Mobile Security | 27 | 2 | 93,1% |
| securiON | OnAV | 26 | 3 | 89,7% |
| Avast | Mobile Security | 24 | 5 | 82,8% |
| AVG | AntiVirus FREE | 24 | 5 | 82,8% |
| Avira | Antivirus Security | 24 | 5 | 82,8% |
| McAfee | Mobile Security | 24 | 5 | 82,8% |
| Protected.Net | Total AV | 24 | 5 | 82,8% |
| AhnLab | V3 Mobile Security | 23 | 6 | 79,3% |
| Ikarus | mobile.security | 23 | 6 | 79,3% |
| LINE | Antivirus | 21 | 8 | 72,4% |
| NortonLifeLock | Norton 360 | 17 | 12 | 58,6% |
| Google | Play Protect | 9 | 20 | 31,0% |

FIGURE 1.27 – Test de détection des logiciels espions pour Android [AV_Test, 2021]

### 1.5.4 Conclusion

Dans ce chapitre, nous avons commencé par mentionner les concepts de base du machine learning et du deep learning. Nous nous sommes concentrés en particulier sur les différents types de réseaux de neurones, après nous avons fait un petit passage au système Android et quelque aspects liés à sa sécurité. Nous avons montré les mécanismes de sécurité soumis par le système android qui sont malheureusement pas parfaits sans l'utilisation d'une application de sécurité étrange, et nous avons conclu ce chapitre par les diffèrent types de malwares mobiles et leurs impacts nocifs sur les appareils mobiles et la vie privée des utilisateurs. Et enfin un aperçu de résultat de test de détection des logiciels espions, qui a révélé les limites de sécurité des systèmes Android, les systèmes les plus populaires au monde entier, et nous a montré la nécessité des autres techniques de détection de malwares qui sont les techniques de détection baser sur le deep learning et qui seront abordés dans le deuxième chapitre.

# CHAPITRE 2

# TECHNIQUES D'ANALYSE ET DE DÉTECTION DES MALWARES SUR LES APPAREILS ANDROID

## 2.1 Introduction

Il existe plusieurs systèmes d'exploitation pour les appareils mobiles, le système Android est le système le plus répandu à l'heure actuelle. Nous mettons donc en évidence ce système et les solutions de sécurité qui doivent être fournies pour protéger l'utilisateur. Ce système fournit une couche initiale de protection qui correspond aux autorisations requises avant d'installer chaque application. Mais c'est plutôt inefficace, car l'utilisateur doit avoir beaucoup de connaissances techniques pour faire la distinction entre les logiciels malveillants et les applications bénignes [Yuan et al., 2016].

Un simple utilisateur a tendance à accorder automatiquement les autorisations requises, d'autant plus qu'il doit choisir entre autoriser toutes les autorisations pour installer l'application ou ne pas l'installer complètement. [Wang et al., 2016].

Dans ce chapitre, nous présentons les recherches les plus importantes dans la détection des malwares pour les appareils Android en utilisant différentes techniques.

## 2.2 Techniques d'analyse des malwares

L'analyse est une étape essentielle pour détecter les malwares. Pour cela , nous devons définir la fonction de ces programmes, et le but de leur développements, afin qu'ils deviennes plus facile à les comprendre par les développeurs en créant un plan de défense contre ces malwares. Les techniques d'analyse des logiciels malveillants se répartissent en trois catégories :

### 2.2.1 Analyse statique

L'analyse statique se concentre sur l'analyse des informations statiques celles obtenues sans exécuter l'application, comme les fichiers exécutables et les codes sources. L'avantage de l'analyse statique est que le sur-coût de calcul est relativement faible et que la vitesse de détection est rapide. L'inconvénient est que les malwares utilisant la technologie d'obscurcissement ne peuvent pas être analysés de manière adéquate par une analyse statique [Feng et al., 2014].

En 2009, Shabtai et al. [Shabtai et al., 2009] ont compilé un ensemble de méthodes de machine learning de détection de malwares, basées sur des caractéristiques statiques extraites d'exécutables. Dans leurs résultats, ils ont conclu que l'application d'algorithmes de pondération (groupe) aux résultats de classification pour les classificateurs individuels améliore considérablement la précision de la classification, comme le montre la figure 2.1.

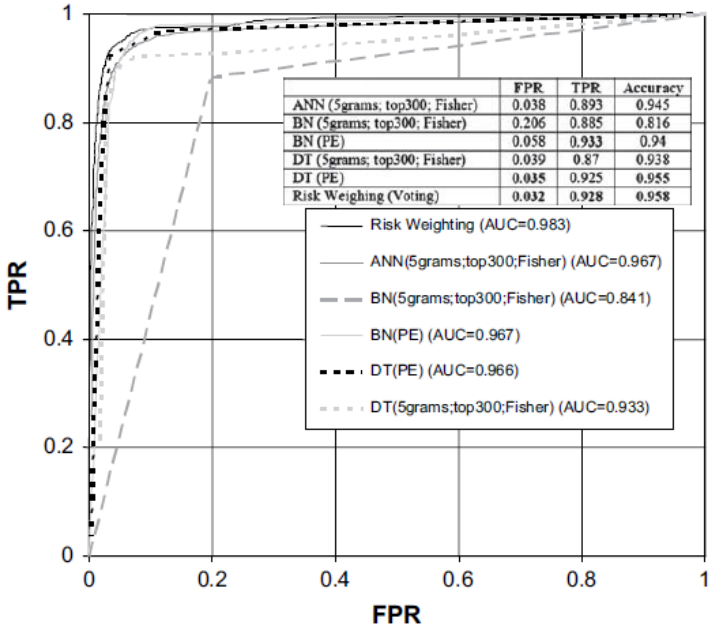


FIGURE 2.1 – Résultats de l'évaluation de cinq classifieur [Shabtai et al., 2009]

En 2016, Fereidooni et al. [Fereidooni et al., 2016] Ont construit un cadre pour la détection basée sur le machine learning, L'importance de leur travail réside dans le développement d'un système de détection de logiciels malveillants léger appelé ANASTASIA pour les smartphones Android. Ils ont utilisé l'analyse statique pour extraire autant de caractéristiques de l'application que possible comme : les autorisations demandées par l'application, le metadata, l'activité Android, etc. Et ce à partir d'un ensemble de données vaste et diversifié (18 677 programmes malignes et 11 187 programmes bénignes). Les résultats obtenus ont montré que la précision du modèle atteint est de 97.3%.

### 2.2.2 Analyse dynamique

L'analyse dynamique consiste à exécuter une application dans un environnement sandbox[1] ou sur un appareil réel. Le système est configuré dans un environnement virtuel fermé et isolé afin que l'échantillon de malware puisse être étudié en profondeur sans risquer d'endommager le système. La surveillance de l'état de l'application lorsqu'elle est en cours d'exécution collecte des informations sur le le comportement de l'application et analyse une série d'informations de données (journal, trafic réseau). Le comportement des applications est généralement surveillé en accédant à des données privées ou en utilisant des appels d'API restreints. L'analyse dynamique peut identifier les comportements malveillants qui ne sont pas détectés par les méthodes d'analyses statiques. L'avantage de l'analyse dynamique est qu'elle peut traiter des malwares utilisant des techniques d'obscurcissement, mais les inconvénients sont que le temps d'analyse et de détection est long et le coût de calcul est élevé [Wong and Lie, 2016].

En 2009, Bayer et al. [Bayer et al., 2009] ont proposé une approche pour collecter et stratifier une variété énormes d'échantillons de malwares présentant un comportement similaire. Ils analysent dynamiquement chaque échantillon malveillant pour identifier ses traces et détecter son comportement. Ensuite, les profils comportementaux sont extraits en extrayant les appels système. L'algorithme de clustering est basé sur le hachage sensible à l'emplacement (LSH), pour résoudre efficacement le problème de clustering en utilisant une approche d'approximation probabiliste. Leurs résultats ont montré une précision élevée de 98% et ont démontré que la méthode proposée est capable d'identifier et de regrouper les logiciels malveillants qui se comportent de manière similaire avec une excellente précision.

En 2017, Bhatia et al. [Bhatia and Kaushal, 2017] ont introduit un moyen d'effectuer une analyse dynamique des applications Android pour les classées. Ils ont développé un système de capture d'appels système qui collecte et extrait les traces d'appels système pour les applications lors des interactions d'exécution. La Figure 2.2 représente leur architecture. Ensuite, les données d'appel système collectées sont analysées pour détecter et catégoriser le comportement des applications Android. Ils se sont appuyés sur une base de données de 50 applications Android bénignes de Google Play Store officiel et de 50 applications malignes obtenues à partir du projet Android Malware Genome, pour analyser leur comportement. Dans leurs expériences, ils ont atteint un niveau de précision de 85% en utilisant un arbre de décision, et un niveau de précision de 88% en utilisant l'algorithme de forêt aléatoire.

### 2.2.3 Analyse hybride

L'analyse hybride combine l'analyse statique et l'analyse dynamique. Dans cette approche, l'application est analysée pour extraire les caractéristiques statiques et dynamiques. Améliorant ainsi la précision de la reconnaissance. Son avantage est qu'elle est plus complète en analyse applicative, elle allie les avantages des deux méthodes d'analyse. Cependant, les inconvénients sont que le temps d'analyse et de détection est long, et l'analyse prend de nombreuses ressources système et la surcharge de calcul est énorme.

La méthode d'analyse hybride peut non seulement améliorer la couverture du code, mais éga-

[1]sandbox c'est un environnement de test isolé, qui permet à l'utilisateur d'exécuter des programmes et d'exécuter des fichiers sans affecter la plate-forme sur laquelle ils sont exécutés [Wong and Lie, 2016].

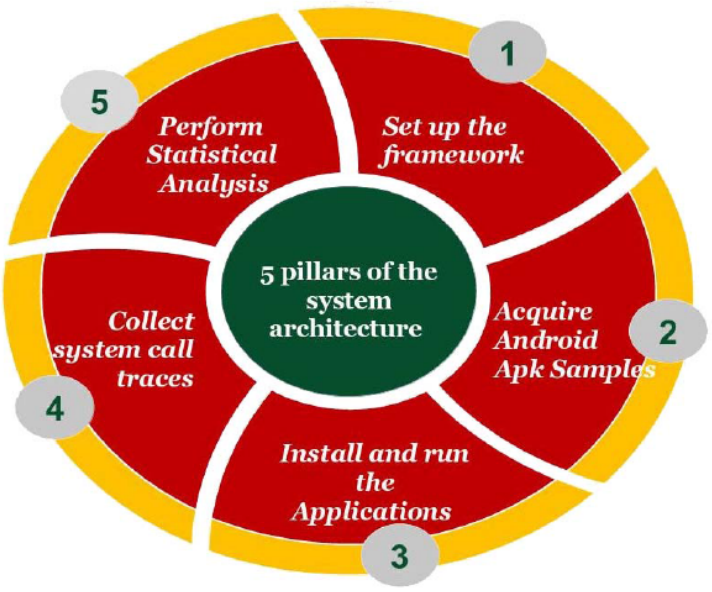


FIGURE 2.2 – L'architecture du système proposé dans [Bhatia and Kaushal, 2017]

lement lutter contre l'obscurcissement ou le cryptage du code. Bien que son coût élevé et sa complexité de mise en œuvre élevée limitant son déploiement, la méthode d'analyse hybride est toujours considérée comme la méthode la plus approfondie et la plus complète [Mahmood et al., 2014, Vidas et al., 2014].

En 2017, Yang et al. [Yang et al., 2017] détectent les malwares dans les appareils Android par une analyse hybride et les classent en fonction d'un algorithme de machine learning. Ensuite, ils ont testé une base de données de 3378 programmes malveillants issus de la plate-forme Drebin [Arp et al., 2014] et de 2140 applications populaires de Google Play [Google_Play, 2021], en utilisant plusieurs algorithmes de machine learning (Random Forest , Support Vector Machine, Naïve Bayes, Logistic Régression , Arbre de décision J48). Random Forest avait la précision de détection la plus élevée a 95.9%.

Une autre recherche récente menée par Chaulagain et al. [Chaulagain et al., 2020] afin de détecter les malewares Android repose sur la technologie d'analyse hybride et d'un modèle d'apprentissage en profondeur dans le but de résoudre le problème de l'interaction humaine (c'est-à-dire sans intervention spécialisée). Ils ont utilisé une base de données de 18 227 échantillons (5 617 applications malignes et 12 610 applications bénignes). En expérimentant plusieurs modèles d'apprentissage en profondeur, ils sont parvenus à une grande précision de 99.98%.

## 2.3 Techniques de détection de malwares

Suite à l'apparition des malwares, les premières techniques qui ont été utilisées pour les détecter dans les appareils mobiles Android, c'étaient celles qui Remarques la consommation de l'énergie et l'utilisation de la batterie de manière anormale. Une application qui utilise au maximum la batterie et les ressources CPU est considérée comme une application malveillante [Zefferer et al., 2013].

Mais ces techniques ne sont pas adaptées au scénario actuel car les appareils mobiles d'aujourd'hui remplissent des fonctions presque équivalentes aux ordinateurs, ils consomment donc pas mal d'énergie [Kim et al., 2008].

Ensuite, l'une des techniques de détection de malwares les plus utilisées est apparue, la détection basée sur les signatures. La signature est une chaîne d'octets qui peut être utilisée pour identifier des malwares spécifiques. Une variété de schémas de correspondance de motifs sont utilisés pour rechercher des signatures [Aycock, 2006].

Les logiciels anti-malwares basés sur les signatures doivent conserver un référentiel de signatures de malwares connus que doit être mis à jour fréquemment lorsque de nouvelles menaces sont détectées. La détection basée sur les signatures est simple, relativement rapide et efficace contre les types de malwares les plus simples. Les principaux problèmes liés à la méthode de détection basée sur les signatures sont les suivants [Borello and Mé, 2008] :

- La détection des signatures nécessite une base de données de signatures mise à jour sinon les logiciels malveillants qui ne se trouvent pas dans la base de données ne sont pas détectés.
- L'extraction et la distribution de signatures est une tâche complexe.
- La création de signature implique une intervention manuelle et nécessite une analyse de code rigoureuse.
- La taille du référentiel de signatures continue de croître à un rythme alarmant.

L'inconvénient majeur est qu'il existe, des techniques d'obscurcissement relativement simples pouvant être utilisées pour éviter la détection de signature.

Concernant ces conditions, les chercheurs ont proposé des méthodes basées sur le machine learning pour détecter les logiciels malveillants. Cette approche évite de créer et de mettre à jour des règles de détection manuellement, ce qui signifie qu'il est possible de détecter les malwares sans extraction de signature.

### 2.3.1 Approches basées sur le machine learning

Au cours de la dernière décennie, l'application des approches de machine learning a considérablement augmenté, notamment en termes de sécurité, en raison de la diversité et de la complexité croissantes des attaques de malwares modernes [Aafer et al., 2013].

Une caractéristique pertinente des approches de machine learning est leur capacité à généraliser, c'est-à-dire la capacité à détecter des attaques inédites ou des variantes inconnues de celles-ci.

Athiq Reheman Mohammed et al. [Mohammed et al., 2019] Ont développé une application qui peut faire la distinction entre un exécutable malveillant et un exécutable légitime à l'aide d'algorithmes de machine learning. Dans cette application, ils s'appuient sur des arbres de décision et des modèles de forêts aléatoires. Leur objectif principal était de créer une application Web utilisant le framework python flask, afin que l'utilisateur puisse télécharger des fichiers exécutables, et du côté serveur, un algorithme de classification de machine learning testerait si les fichiers téléchargés sont malveillant ou non, et le résultat serait transmit à l'utilisateur.

L'ensemble de données a été extrait de Kaggle et se compose de paramètres de fichiers binaires exécutables légitimes et malveillants. Il se compose de 138 048 exemples d'apprentissage et 54 caractéristiques. Dans leur expérience, ils ont utilisé des algorithmes de classification basés sur des arbres, à savoir les arbres de décisions et les forêts aléatoires comme classifieurs et la précision de chacun a été comparée. Après l'expérimentation, la précision du modèle était de 98,9 % pour l'arbre de décision et de 99.4 % pour les forêts aléatoires.

Mais ces résultats ne peuvent pas être généralisés, car ils ne dépendent que de l'ensemble de données spécifique.

Barreno et al. [Barreno et al., 2006, Barreno et al., 2010] Stipulent que les algorithmes de machine learning sont construits sur l'hypothèse que les données d'entraînement et de test suivent la même distribution de probabilité de base, ce qui les rend vulnérables aux attaques bien faites qui violent cette hypothèse. Cela signifie que le machine learning lui-même peut être le maillon le plus faible de la chaîne de sécurité. Des travaux ultérieurs ont confirmé cette intuition, montrant que les techniques de machine learning peuvent être grandement affectées par des attaques soigneusement conçues qui exploitent la connaissance de l'algorithme d'apprentissage, par exemple, des attaquants qualifiés peuvent manipuler les données au moment du test pour éviter la détection, ou injecter des échantillons de poison dans les données d'apprentissage pour induire en erreur l'algorithme d'apprentissage et ainsi provoquer des erreurs de classification [Biggio et al., 2012, Biggio et al., 2013].

Ramakrishna et al. [Ramakrishna et al., 2021] ont proposé un système utilisant des algorithmes de machine learning pour distinguer si une certaine partie d'un document/programme est affectée ou non par des malwares. Ils ont analysé les données et les ont classées en fonction de leurs caractéristiques en groupes de neuf familles. Ces familles contiennent un ensemble de données d'environ un demi-téraoctet. Ils se concentrent principalement sur la classification des malwares en fonction des statistiques de propriétés statiques du fichier binaire et de l'assembleur, et au lieu d'utiliser le clustering, ils utilisent le multi-threading, de sorte que cette méthode puisse fournir des performances dans une fonctionnalité limitée et une tâche de calcul faible, et maintenir le compromis entre la complexité et les performances. La Figure 2.3 illustre la conception du système proposé. Ils ont comparé ensuite les quatre algorithmes, qui sont : K-Nearest Neighbor, Logistic Regression, Random Forest, et XGboost. En utilisant la technique de fusion de combinaison de fonctionnalités, XGboost a atteint une précision de modèle très élevée parmi tous les autres algorithmes, soit une précision de 99,83 % avec une valeur de perte de 0,01 %.

Dans leur travail, ils ont introduit un cadre de classification des logiciels malveillants qui est moins complexe dans l'extraction et la classification des fonctionnalités, mais peut avoir un impact négatif sur la généralisation de leur modèle en raison de leur dépendance à l'analyse statique uniquement.

### 2.3.2 Approches basées sur le deep learning

Les techniques basées sur le machine learning peuvent détecter des logiciels malveillants modernes et obscurcis avec une grande précision, mais le problème majeur dans les algorithmes de machine learning, c'est l'obligation de l'intervention des experts dans l'extraction des caractéristiques pour réduire la complexité des données utilisées. C'est un processus généralement

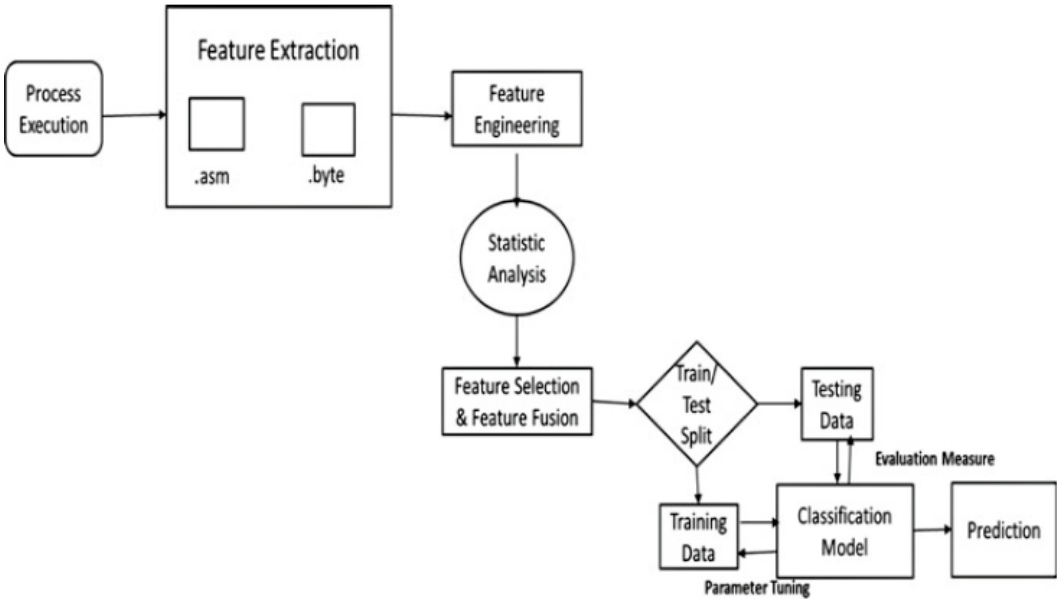


FIGURE 2.3 – Conception du système proposé [Ramakrishna et al., 2021]

difficile et coûteux en termes de temps et d'expertise. Ces dernières années, un nouveau domaine connu sous le nom de deep learning a émergé, et qui a montré des résultats prometteurs et convaincants dans plusieurs domaines, tels que le traitement d'images et le langage naturel. Par conséquent, plusieurs études sont apparues qui utilisent le deep learning pour détecter les logiciels malveillants dans le système Android en raison des bons résultats obtenus par cette approche [Liu et al., 2021].

En 2014, Zhenlong Yuan et al. [Yuan et al., 2014] ont proposé un modèle de deep learning, puis le comparé avec d'autres techniques de machine learning. Premièrement, ils ont extrait plus de 200 caractéristiques d'analyse statique et d'analyse dynamique de l'application de détection de malwares Android. Deuxièmement, ils se sont appuyés sur un algorithme d'entraînement semi-supervisé, qui se compose de deux phases, la phase de pré-apprentissage non supervisée «unsupervised pre-training phase» et la phase de rétro-propagation supervisée «supervised back-propagation phase ». En pré-apprentissage, ils se sont appuyés sur le Deep Belief Network (DBN) [Bengio, 2009] pour le pré-apprentissage, ce qui est utile pour une meilleure description des applications Android. Dans l'étape de rétro-propagation, le réseau de neurones préalablement formé est affiné à la valeur spécifiée de manière supervisée. Après cela, tout le modèle d'apprentissage en profondeur est construit. Ils ont expérimenté des groupes d'applications génériques (un mélange d'applications bénignes et malignes). Collection de malwares (250 échantillons) de contagio mobile et collection d'applications bénignes des 250 meilleures applications du Google Play Store. En comparant le modèle de deep learning proposé avec les cinq autres modèles de machine learning (SVM, C4.5, Naive Bayes, LR, MLP), le modèle a atteint un niveau de précision élevé avec plus de 96%. Comme indiqué dans la Table 2.1

Leur modèle offre une excellente précision, d'autant plus qu'ils s'appuient à la fois sur l'analyse statique et l'analyse dynamique, mais nous notons que l'ensemble de données utilisé pour l'entraînement est relativement petit, Pour généraliser le modèle, il doit être entraîné sur un ensemble de données plus important.

TABLE 2.1 – Comparaison des résultats des modèles [Yuan et al., 2014]

| Model | Training Set | Test Set | Accuracy |
|---|---|---|---|
| SVM (Support vector machine) | 300 | 200 | 80.0% |
| C4.5 (Decision tree algorithm) | 300 | 200 | 77.5% |
| Naive Bayes | 300 | 200 | 79.0% |
| LR (Logistic regression) | 300 | 200 | 78.0% |
| MLP (Multilayer Perceptron) | 300 | 200 | 79.5% |
| **DL (Deep learning)** | 300 | 200 | 96.5% |

Il existe plusieurs études qui utilisent un réseau de neurones convolutifs (CNN), pour détecter les malwares, y compris l'étude de Kuo et al. [Kuo and Lin, 2018] qui Ont proposées dans leur article une méthode de détection des malwares Android basée sur les Réseau de neurones convolutifs, en utilisant une technique d'analyse statique. Ils ont rencontré un problème que lors de la technique de décompilation du fichier apk le code source ne soit pas récupéré correctement et complètement. Ensuite, ce fichier est converti en une image, puis cette image est introduite dans le modèle cnn pour la formation du modèle, et à l'aide de ce modèle formé, il détermine si le code malveillant est présent dans le fichier APK ou non. Le processus de détection est divisé en trois parties : l'extraction du fichier classes.dex, la création d'un ensemble de données et la découverte du fichier APK, comme le montrent les figures 2.4,2.5 et 2.6, respectivement. Ils ont utilisé une base de données de 1 166 Exemples d'APK, dont 523 échantillons bénignes de Google Play [Google_Play, 2021] et 643 échantillons malignes de VirusShare [Virus_Share, 2021] et du National Center for High Performance Computing [National_Center, 2021]. Dans leur expérience, ils ont comparé la précision du modèle avec différents paramètres tels que le nombre d'époques et le nombre de neurones, et ils ont obtenu une précision de 94 %.

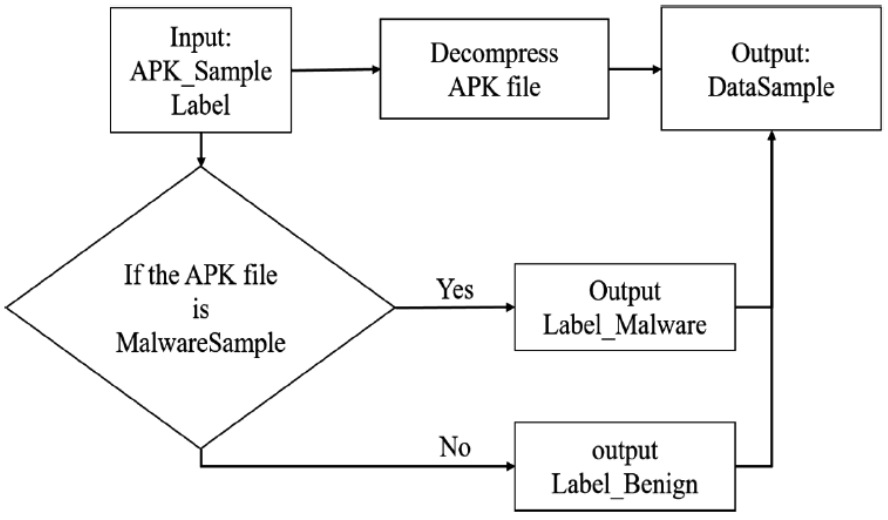


FIGURE 2.4 – Extraire le fichier classes.dex [Kuo and Lin, 2018]

Sourav et al. [Sourav et al., 2019] Ont proposé un modèle de deep learning basé sur un réseau de neurones artificiels, pour prédire les malwares dans les applications Android. Après avoir lu

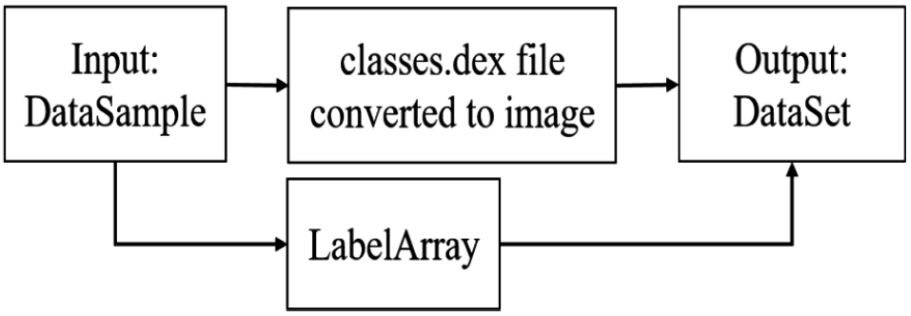


FIGURE 2.5 – Créer un ensemble de données [Kuo and Lin, 2018]

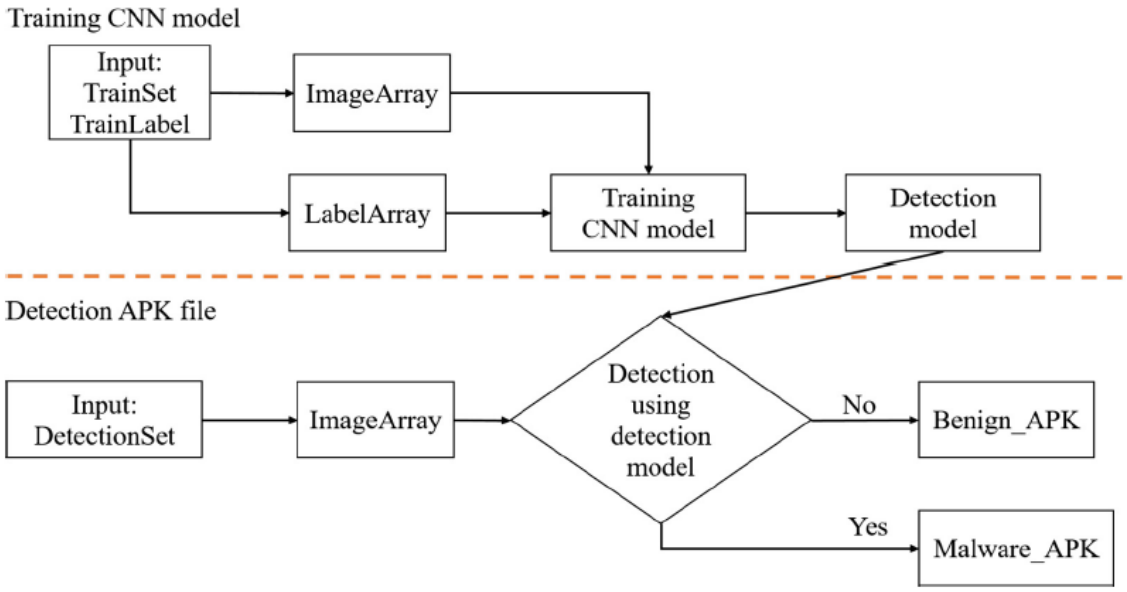


FIGURE 2.6 – Module de détection [Kuo and Lin, 2018]

l'ensemble de données, qu'il s'agisse de logiciels malveillants ou d'applications bénignes, Ils ont identifié des caractéristiques utiles à l'aide de deux techniques distinctes : analyse composantes principales (ACP) et Extra Tree Classifier en séparant les caractéristiques importantes. Les caractéristiques non pertinentes sont ensuite supprimées pour une meilleure généralisation et précision. Ensuite, ils ont utilisé l'ensemble de données de Kaggle, l'ensemble de données contenant 338 applications classées comme « bénignes » ou « malignes ». Ils utilisent androguard, un projet open source pour extraire les caractéristiques des fichiers apk, mettant constamment à jour l'ensemble d'entraînement avec de nouveaux fichiers malveillants pour améliorer le classificateur.cela résout le déséquilibre entre les applications bénignes et malignes en maintenant le rapport entre les entrées bénignes et malignes égal. Ils ont utilisé la régression logistique, Naïve Bayes et les arbres améliorés par gradient avec un réseau de neurones artificiels et ont comparé ses performances sur les données de test. Après leur expérimentation et en comparaison avec d'autres modèles, l'ANN a montré une meilleure précision de classification que toutes les autres techniques, atteignant 95%, la Figure 2.7 illustre cela.

Leur modèle proposé peut être relativement précis en raison de la petite taille des données applicables, car il doit être formé sur un ensemble de données plus important.

| Name | Accuracy(%) |
|---|---|
| Naive Bayes | 88.3 |
| Random Forests | 90.8 |
| Gradient Boosting | 91 |
| Logistic Regression | 93.6 |
| Attention Based ANN | 95.3 |

FIGURE 2.7 – Précision de la classification entre les modèles [Sourav et al., 2019]

Il existe plusieurs recherches utilisant l'approche de réseaux de neurones récurrents (RNN), y compris ce qui a été fait par William Younghoo Lee et al. [Lee et al., 2019] le modèle reconnaît l'association généralisée entre La chaîne ambiguë du nom du package d'application et du nom du développeur. ils ont prouvé que le modèle proposé est robuste contre l'obscurcissement, suffisamment évolutif et léger pour être déployé sur les appareils Android en tant qu'application de détection de malware efficaces.

## 2.4 Outils de détections de Malware

Les applications antivirus Android sont très importantes pour éviter le risque d'être infecté par des applications corrompues et d'autres types de logiciels malveillants. Les meilleures applications antivirus Android offrent non seulement la détection et la prévention des logiciels malveillants, mais également une multitude de fonctionnalités de confidentialité et d'antivol. Certaines d'entre elles ont des versions gratuites et d'autres sont payantes. Les plus importantes de ces applications sont décrites dans ce qui suit.

### 2.4.1 Bitdefender Mobile Security

Bitdefender fournit une puissante couche de protection pour les appareils Android. Cette application a un léger impact sur les performances de l'appareil, dispose d'un bloqueur de sites Web malveillants et d'un client VPN, et offre également une excellente intégration avec la montre d'Android Wear. Il offre également des outils puissants pour la protection de la vie privée, notamment le verrouillage des applications, un scanner Wi-Fi, des fonctions antivol et des notifications de violation de données [Android_Antivirus, 2021].

### 2.4.2 Norton Mobile Security

Norton Mobile Security est l'une des meilleures applications anti-malware pour Android. Il a une fonction pour vérifier l'utilisation intensive des données et le comportement inhabituel sur les appareils. Il analyse également les applications de Google Play Store et recherche les risques liée à la sécurité et à la confidentialité avant même de les installer, mais il lui manque certains

éléments de sécurité de base tels que l'antivol, le blocage des liens malveillants et le verrouillage des applications individuelles [Android_Antivirus, 2021].

### 2.4.3 Avast Mobile Security

Avast Mobile Security et Antivirus est l'une des applications antivirus Android les plus populaires, offrant de nombreuses fonctionnalités de protection. Mais si la protection contre les malwares d'Avast est bonne, elle n'est pas sans défauts. Comme les publicités gênantes dans la version gratuite, et sa faiblesse dans les fonctions antivol et de blocage d'appels [Android_Antivirus, 2021].

### 2.4.4 Kaspersky Mobile Antivirus

Kaspersky Mobile Antivirus, également connu sous le nom de Kaspersky Internet Security pour Android, offre une protection contre les logiciels malveillants presque parfaite, un faible impact sur le système et un excellent bloqueur d'appels. Il n'y a pas d'annonces dans la version gratuite, les utilisateurs de la version gratuite bénéficient de plusieurs fonctions telles que le filtrage des appels, la prise en charge d'Android Wear et un ensemble puissant de fonctions antivol, mais le seul inconvénient est de scanner chaque nouvelle application manuellement. Les utilisateurs qui paient les frais bénéficient d'analyses automatiques des nouvelles applications, du verrouillage des applications et du blocage des sites d'hameçonnage [Android_Antivirus, 2021].

### 2.4.5 McAfee Mobile Security

McAfee offre une excellente protection contre les logiciels malveillants et une pléthore de fonctionnalités utiles telles qu'une fonctionnalité « invité » qui permet à d'autres personnes d'utiliser votre téléphone en toute sécurité pendant une courte période. Dans la version gratuite, l'utilisateur bénéficie de fonctionnalités antivol, d'un moyen de suivre l'utilisation des données pour chaque application et d'un scanner de sécurité Wi-Fi. La version regorge également d'annonces et de suggestions redondantes. La version payante offre un verrouillage d'application, un vérificateur d'URL, une assistance technique 24h/24 et 7j/7 et plus aucune publicité [Android_Antivirus, 2021].

## 2.5 Datasets

L'ensemble de données est un élément fondamental pour la détection des malwares, car il joue un rôle clé dans la détermination de la précision et de la généralisabilité du modèle proposé. Il existe une grande variété d'ensembles de données, et les plus célèbres et les plus importants de ces groupes sont :

### 2.5.1 Genome

L'un des ensembles de données les plus célèbres est le genome, qui a été utilisé dans de nombreuses recherches afin de détecter les logiciels malveillants Android. Il contient 1260 échantillons couvrant la majorité des familles de malwares, de sa première apparition en août 2010 à sa dernière en octobre 2011. Mais depuis le 21/12/2015, les auteurs de génomes ont cessé de prendre en charge le partage d'ensembles de données en raison de ressources limitées et d'un changement important dans les derniers logiciels malveillants [Wei et al., 2017] [Zhou and Jiang, 2012].

### 2.5.2 Drebin

Drebin effectue une analyse statique à grande échelle, collectant autant de caractéristique d'application que possible. Ces données sont contenues dans un espace vectoriel commun, de sorte que les modèles typiques qui indiquent les logiciels malveillants peuvent être automatiquement identifiés. Drebin présente 123453 applications bénignes de différents marchés et 5560 applications de 179 familles d'applications malignes différentes [Arp et al., 2014].

### 2.5.3 Malgenome-215

Malgenome-215 se compose d'un vecteurs de caractéristiques de 3799 échantillons d'applications (2539 échantillons bénins et 1260 échantillons d'applications malignes), du projet Android Malware Genome. Cet ensemble d'échantillons de malwares a été largement utilisé par la communauté de recherche sur les logiciels malveillants pour ses 215 fonctionnalités [Zhou and Jiang, 2012].

### 2.5.4 Yerima dataset

Un autre jeu de données développé par Yerima [Yerima, 2018]. L'ensemble de données contient 215 traits avec et 15036 échantillons (5560 sont des applications malignes et 9476 sont des applications bénignes).

### 2.5.5 CCCS-CIC-AndMal-2020

Cette collection contient un énorme nouvel ensemble de données sur les logiciels malveillants android, connus sous le nom de CCCS-CIC-AndMal -2020, c'est un extraits d'analyses statiques et dynamiques [CCCS-CIC-AndMal, 2020]. Dans le prochain chapitre, nous baserons nos expérimentations sur ces données.

## 2.6 Conclusion

Dans ce chapitre, nous avons donné un aperçu des techniques utilisées pour analyser et détecter les malwares dans les appareils Android, car nous avons mentionné un certain nombre de recherches qui ont traité cette tâche, en commençant par les premières techniques, en passant par les techniques basées sur le machine learning, et enfin les techniques de deep learning, les récentes techniques de deep learning ont montré de très bons résultats, sur lesquels nous nous concentrerons et utiliserons pour construire un modèle de détection de malwares dans notre prochain chapitre. Nous avons également donné un aperçu des outils de détection de malwares les plus populaires et les plus importants, et de certains ensembles de données (dataset) utilisés pour la détections des malwares.

# CHAPITRE 3

# IMPLÉMENTATION ET EXPÉRIMENTATION

## 3.1 Introduction

Dans ce chapitre, nous présentons notre expérience de construction d'un modèle de deep learning pour la détection des malwares. Dans notre modèle, nous utilisons une architecture basée sur deep neural network (DNN). Nous décrivons d'abord l'architecture du réseau proposé, puis l'environnement et le jeu de données sur lesquels nous avons travaillé, puis nous passons à la présentation des étapes de mise en œuvre, et en fin nous discutons les résultats obtenus.

## 3.2 Architecture

Dans cette expérience, nous avons utilisé un réseau de deep learning (DNN) , la figure suivante 3.1 montre l'architecture de ce réseau. Elle comporte trois couches principales :

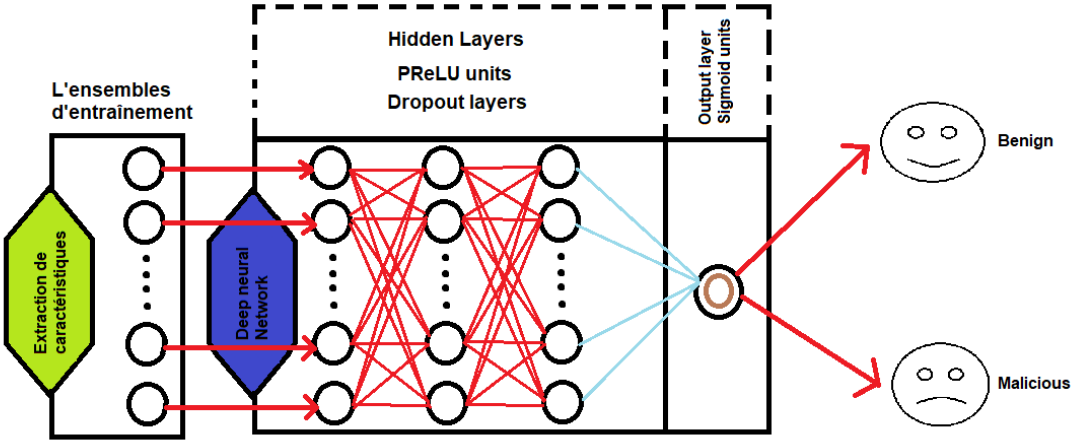


FIGURE 3.1 – L'architecture DNN proposée

**Couche d'entrée** La couche d'entrée est composée de 500 entrées, ces entrées expriment les caractéristiques et les attribues qui déterminent chaque application, ces attributs sont extraites de l'analyses statiques et dynamiques des données d'apprentissages.

**Couche cachée** Elle contient plusieurs couches, 2, 3, 4, 5 ou 6 couches, chaque couche dense contient des fonctions d'activations Relu, chaque deux couches sont suivies d'une couche dropout, ajoutées pour réduire le sur-apprentissage du modèle.

**Couche de sortie** Elle se compose d'un seul neurone, qui représente l'unité sigmoïde pour classer les applications.

## 3.3 Environnement

En raison de la faible efficacité de nos ordinateurs, nous avons utilisé google colab pour développer notre modèle.

**Google Colab** : nous l'utilisons pour identifier et classer l'ensemble de données et pour créer un modèle d'entraînement en utilisant les propriétés suivantes :

- RAM : 12.75GB
- Espace disque : 25GB
- CPU : Intel(R) Xeon(R)
- CPU Freq : 2.30GHz
- No. CPU Cores : 2
- Famille de CPU : Haswell
- GPU : Nvidia K80 / T4
- Mémoire GPU : 12GB/16GB

Nous écrivons notre code en utilisant : le langage de programmation python3 sous jupyter notebook et en utilisant les bibliothèques suivantes :

- **Os**

Le système d'exploitation fait partie des modules utilitaires standard de Python. Ce module fournit un moyen portable d'utiliser les fonctionnalités dépendantes du système d'exploitation. Le module *os* inclut de nombreuses fonctions pour interagir avec le système de fichiers.

- **Matplotlib**

Est une bibliothèque de visualisation de données multiplateforme complète, construite sur des tableaux NumPy.

- **Numpy**

Est une bibliothèque de calcul scientifique qui prend en charge de grands tableaux et matrices multidimensionnels avec une grande collection de fonctions mathématiques de haut niveau pour opérer sur ces tableaux.

- **Pandas**

Pandas est une bibliothèque logicielle écrite pour le langage de programmation Python pour la manipulation et l'analyse de données. En particulier, il propose des structures de données et des opérations pour manipuler des tableaux numériques et des séries chronologiques.

- **LabelEncoder**

Cet encodeur fait partie de la bibliothèque SciKit-learn et est utilisé pour convertir du texte ou des données catégorielles en données numériques auxquelles le modèle s'attend et avec lequel il fonctionne mieux.

- **Tensorflow**

TensorFlow est une bibliothèque Python pour le calcul numérique rapide créée et publiée par Google.

Il s'agit d'une bibliothèque de base qui peut être utilisée pour créer des modèles Deep Learning directement ou en utilisant des bibliothèques d'encapsulation qui simplifient le processus construit sur TensorFlow.

- **Keras**

Est une bibliothèque de réseaux de neurones conçue pour permettre une expérimentation rapide avec des réseaux de neurones profonds et elle est prise en charge dans la bibliothèque principale TensorFlow depuis 2017.

- **Csv**

Est une bibliothèque qui implémente des classes pour lire et écrire des données tabulaires au format CSV.

## 3.4 Expérimentation

### 3.4.1 Dataset

Dans ce projet, nous avons utilisé un ensemble de données appelé CCCS-CIC-AndMal-2020 (un projet de l'Institut canadienne de la cybersécurité (CIC) en collaboration avec le Centre canadien pour la cybersécurité (CCCS)). [CCCS-CIC-AndMal, 2020] [Keyes et al., 2021,Rahali et al., 2020].

Cette collection contient un sous-ensemble de données massif et nouveau de logiciels malveillants Android. L'ensemble de données total comprend 200 000 échantillons de logiciels bénigne et 200 000 échantillons de logiciels malignes totalisant 400 000 applications Android avec 14 catégories de logiciels malveillants (adware, backdoor, file infector, no category, Potentially Unwanted Apps (PUA), ransomware, riskware, scareware, trojan, trojan-banker, trojan-dropper, trojan-sms, trojan-spy and zero-day) et 191 familles de logiciels malveillants importantes.

Le mérite revient au Centre canadien pour la cybersécurité (CCCS) d'aider l'Institut CIC à capturer les applications malveillantes Android du monde réel pour les analyser. L'Institut

CIC a utilisé un service d'analyse des virus et des logiciels malveillants appelé VirusTotal, pour spécifier les familles des malwares et étiqueter l'ensemble de données qui sont examinés par une dizaine d'antivirus pour incorporer la fiabilité dans ces données.

La Table 3.1 présente le détail de 14 catégories de logiciels malveillants Android ainsi que le nombre des familles et d'échantillons impliqués dans l'ensemble de données.

TABLE 3.1 – Catégories de logiciels malveillants Android

| **Catégorie** | **Nombre de familles** | **Nombre d'échantillons** |
|---|---|---|
| Adware | 48 | 47210 |
| Backdoor | 11 | 1538 |
| File Infector | 5 | 669 |
| No Category | - | 2296 |
| PUA | 8 | 2051 |
| Ransomware | 8 | 6202 |
| Riskware | 21 | 97349 |
| Scareware | 3 | 1556 |
| Trojan | 45 | 13559 |
| Trojan-Banker | 11 | 887 |
| Trojan-Dropper | 9 | 2302 |
| Trojan-SMS | 11 | 3125 |
| Trojan-Spy | 11 | 3540 |
| Zero-day | 8 | 13340 |

Dans la Table 3.2, nous montrons les familles d'une seule catégorie de malwares et le nombre d'échantillons capturés, à titre d'exemple on prend File Infector.

TABLE 3.2 – Les familles de (File Infector) avec le nombre d'échantillons détectés

| **No** | **Famille** | **Nombre d'échantillons capturés** |
|---|---|---|
| 1 | commplat | 77 |
| 2 | leech | 99 |
| 3 | tachi | 45 |
| 4 | gudex | 14 |
| 5 | aqplay | 407 |

L'Institut CIC a utilisé l'ensemble de données Androzoo comme une source pour les applications Android bénignes. Androzoo, contient actuellement plus de huit millions d'applications Android

collecter à partir de différentes sources, notamment le marché Android officiel, Google Play, Anshi, AppChina, 1mobile et l'ensemble de données du projet Genome. Chaque semaine Une liste est créé par le CIC contenant toutes les informations détaillées sur les applications Android bénignes.

### Analyse statique

De nombreuses caractéristiques pouvant être utilisées dans l'analyse statique des applications, qui ont été extraites du fichier AndroidManifest.xml (un fichier qui fournit les infos essentielles des applications aux outils d'Android), les caractéristiques principales utilisées sont :

- **Activités :** Une activité Android est un écran de l'interface utilisateur de l'application Android.
- **Récepteurs de diffusion et fournisseurs de contenu :** Les Récepteurs gèrent la communication entre l' OS et les applications et les fournisseurs gèrant les problèmes de gestion des données.
- **Metadata :** Il s'agit essentiellement d'une option supplémentaire pour stocker des informations accessibles tout au long du projet.
- **Les autorisations demandées par l'application :** Elle protège la vie privée de l'utilisateur et est nécessaire pour accéder aux données sensibles de l'utilisateur (telles que les contacts et les SMS).
- **Fonctionnalités du système** Telles que la caméra et Internet.

### Analyse dynamique

Après l'exécution des malwares dans des environnements émulés, six catégories de fonctionnalités sont extraites qui représente les changements de comportement des catégories et familles de malware :

- **Mémoire :** Les caractéristiques de mémoire définissent les activités effectuées par les logiciels malveillants en utilisant la mémoire.
- **API :** Les caractéristiques de l'interface de programmation d'applications (API) délimitent la communication entre deux applications.
- **Réseau :** Les caractéristiques du réseau décrivent les données transmises et reçues entre d'autres appareils du réseau.
- **Batterie :** Les caractéristiques de la batterie décrivent l'accès au verrouillage de la batterie et aux services par les malwares.
- **Logcat :** Les caractéristiques de Logcat écrivent des messages de journal correspondant à une fonction exécutée par un malware.
- **Process :** Les caractéristiques de processus comptent l'interaction des logiciels malveillants avec le nombre total de processus.

Après avoir téléchargé l'ensemble de données d'entrainement à partir du site du constructeur, nous avons regroupé les fichiers téléchargés (14 fichiers malwares et 5 fichiers bénignes) en

un seul fichier, après nous avons transféré le fichier vers Google Drive. Nous avons pris la totalité des échantillons (200000 malignes et 200000 bénignes). Chaque échantillon possède 500 caractéristiques (extraites d'analyses statiques et dynamiques), Nous avons partitionné l'ensemble de données en deux parties, 80% pour l'entraînement (320000 échantillons) et 20% pour tester le modèle (80000 échantillons) à l'aide de la technique du Train-Test Split.

### 3.4.2 Résultats et discussion

Pour une meilleure exploitation de l'ensemble de données, nous avons expérimenté des modèles avec différentes combinaisons de couches cachées. Tous les modèles ont été entraînés sur les 320000 échantillons, et validés et évalués sur les 80000 échantillons de test. La fonction de perte utilisée est binary-crossentropy, et en ce qui concerne l'optimisation, on a utilisés l'optimiseur d'Adam, nous avons testé chaque fois leur taux de précision et de perte pour choisir le meilleur modèle de détection.

TABLE 3.3 – Résultats d'apprentissage en profondeur avec différentes combinaisons de couches cachées

| No. de couches | No. de neurones | TPR | TNR | FPR | FNR | Accuracy | Perte | Temps (min :sec) |
|---|---|---|---|---|---|---|---|---|
| 2 | [32,16] | 0.9614 | 0.9791 | 0.0208 | 0.0385 | 0.9709 | 0.0821 | 02 :22 |
| 3 | [64,32,16] | 0.9672 | 0.9813 | 0.0186 | 0.0327 | 0.9748 | 0.0729 | 03 :34 |
| 3 | [200,100,200] | 0.9583 | 0.9879 | 0.0141 | 0.0416 | 0.9740 | 0.0776 | 06 :17 |
| **3** | **[200,200,200]** | **0.9714** | **0.9804** | **0.0195** | **0.0285** | **0.9763** | **0.0755** | **07 :23** |
| 4 | [128,64,32,16] | 0.9605 | 0.9856 | 0.0143 | 0.0394 | 0.9739 | 0.0739 | 04 :21 |
| 4 | [300,200,100,50] | 0.9687 | 0.9822 | 0.0177 | 0.0312 | 0.9760 | 0.0776 | 09 :23 |
| 5 | [256,128,64,32,16] | 0.9683 | 0.9815 | 0.0184 | 0.0316 | 0.9754 | 0.0757 | 06 :14 |
| 6 | [300,200,100,50,25,10] | 0.9707 | 0.9810 | 0.0190 | 0.0293 | 0.9762 | 0.0737 | 08 :21 |

La Table 3.3 montre les résultats des expériences menées pour évaluer les performances de l'approche Deep Learning avec différentes combinaisons de couches cachées. huit ensembles différents de neurones cachés, contenant 2, 3, 4, 5 et 6 couches cachés, ont été appliqués afin d'atteindre les meilleures performances possibles en fonction de la précision. Les résultats de ce tableau montrent que l'association [200,200,200] atteint les meilleures performances par rapport aux autres formulations, avec un temps opératoire de sept minutes et 23 secondes.

Nous pouvons voir que notre modèle peut atteindre une précision de 0,9763 en fixant le nombre de couches à 3 et en sélectionnant des neurones suivant la formule mentionné précédemment. La Figure 3.2 montre ce modèle d'apprentissage en profondeur :

La Figure 3.3 montre les changements de précision de ce modèle en termes d'époques, où :

- De 0 à 5 époques, nous constatons une progression rapide de la précision de train_accuracy et test_accuracy.
- De 5 à 25 époques, on remarque que le train_accuracy progresse lentement, jusqu'à ce qu'il se stabilise dans la plage entre 97 et 98%, quant à test_accuracy, la précision est plus de 97%.

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_11 (Dense)             (None, 200)               100200
_________________________________________________________________
dense_12 (Dense)             (None, 200)               40200
_________________________________________________________________
dropout_4 (Dropout)          (None, 200)               0
_________________________________________________________________
dense_13 (Dense)             (None, 200)               40200
_________________________________________________________________
dense_14 (Dense)             (None, 1)                 201
=================================================================
Total params: 180,801
Trainable params: 180,801
Non-trainable params: 0
_________________________________________________________________
```

FIGURE 3.2 – modèle d'apprentissage en profondeur

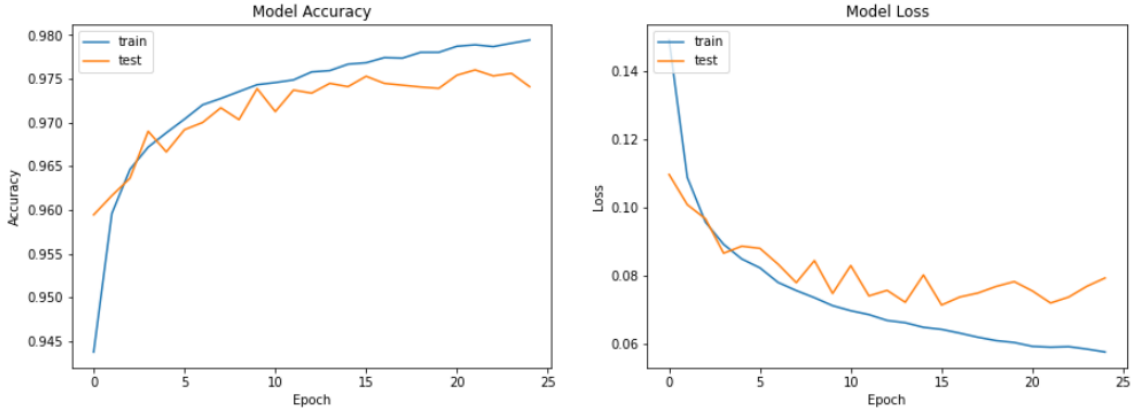


FIGURE 3.3 – Précision du modèle d'apprentissage en profondeur

## 3.5 Conclusion

Dans ce chapitre, nous avons présenté le meilleur modèle que nous avons expérimenté, avec une grande précision de 97.63% obtenue à partir d'un modèle de trois couches cachées entraînées sur l'ensemble de données CCCS-CIC-AndMal- 2020. Ceci est principalement dû aux nombreuses caractéristiques obtenues à partir de l'analyse statiques et l'analyse dynamique de l'ensemble de données.

La précision du modèle peut être améliorée en utilisant plus de données ,mais les capacités offertes par google colab sont limitées.

# CONCLUSION GÉNÉRALE

Le célèbre système mobile Android est exposé à d'innombrables malwares qui sont dissimulés dans un grand nombre d'applications bénignes sur les boutiques d'applications Android, et qui menacent sérieusement la sécurité d'Android.

Dans ce mémoire, nous avons discuté des différents types de techniques de détection de malwares Android à l'aide de diverses méthodes de deep learning. Nous nous sommes focalisés dans notre étude sur la détection de malwares Android à l'aide d'un réseau de neurones feedforward multicouches.

Notre contribution consiste à créer un réseau de neurones de type deep neural network (DNN), pour entrainer un ensemble de données énormes appelé (CCCS-CIC-AndMal-2020) qui contient une collection d'applications nouvelles et variées de 400 000 échantillons, diviser entre bénignes et malignes, l'ensemble de données est prétraitées à travers une analyse statique et une autre analyse dynamique. L'entrainement de l'ensemble de données été fait à travers diverses combinaisons de couches cachées, après le changement subséquent de nombre de couches cachées, nous avons remarqué qu'avec une association de neurones de [200,200,200] et trois couches cachées, le modèle a atteint 97.63% de précision, et c'est la meilleure précision par rapport aux autres formulations, et dans un temps opératoire raisonnable de moins de huit minutes.

Les résultats obtenus sont satisfaisants et auraient pu être améliorés si aucune limitation n'est imposée par Google Colab. Afin d'obtenir des résultats plus fiables et plus généraux pour notre problème de détection de malwares Androïd, nous suggérons de tester ce modèle sur d'autres bases de données, et utiliser une combinaison variété d'algorithmes de deep learning tels que des CNN et des RNN.

Au cours de notre travail dans ce mémoire, nous avons découvert la façon de réaliser un projet, nous avons appris à savoir chercher, à savoir traiter les problèmes, et surtout à travailler collectivement et à s'entraider pour être capable et atteindre notre but.

# BIBLIOGRAPHIE


Aafer, Y., Du, W., and Yin, H. (2013). Droidapiminer : Mining api-level features for robust malware detection in android. In *International conference on security and privacy in communication systems*, pages 86–103. Springer.

Android (2021). Système android. `https://www.android.com/gms/`. Consulté en septembre 2021.

Android_Antivirus (2021). Best android antivirus. `https://www.tomsguide.com/best-picks/best-android-antivirus`. Consulté en juillet 2021.

Android_Platform (2021). Android platform architecture. `https://developer.android.com/guide/platform#linux-kernel`. Consulté en juillet 2021.

Android_Safety (2021). Designed for your safety. `https://www.android.com/safety/`. Consulté en septembre 2021.

Android_Security (2020). Android security. `https://source.android.com/security?hl=en`. Consulté en septembre 2021.

Arp, D., Spreitzenbarth, M., Hubner, M., Gascon, H., Rieck, K., and Siemens, C. (2014). Drebin : Effective and explainable detection of android malware in your pocket. In *Ndss*, volume 14, pages 23–26.

AV_Test (2021). Stalkerware for spying under android. `https://www.av-test.org/en/news/stopped-in-its-tracks-stalkerware-for-spying-under-android/`. Consulté en septembre 2021.

AV_Test_Institute (2021). About the av-test institute. `https://www.av-test.org/en/about-the-institute/`. Consulté en septembre 2021.

Aycock, J. (2006). *Computer viruses and malware*, volume 22. Springer Science & Business Media.

Ayodele, T. O. (2010). Types of machine learning algorithms, new advances in machine learning, yagang zhang (ed.), intech, 2010, doi : 10.5772/9385. *Availablefrom : http ://www. intechopen. com/books/new-advances-in-machinelearning/types-of-machinelearning-algorithms*.

Barreno, M., Nelson, B., Joseph, A. D., and Tygar, J. D. (2010). The security of machine learning. *Machine Learning*, 81(2) :121–148.

Barreno, M., Nelson, B., Sears, R., Joseph, A. D., and Tygar, J. D. (2006). Can machine learning be secure ? In *Proceedings of the 2006 ACM Symposium on Information, computer and communications security*, pages 16–25.

Bayer, U., Comparetti, P. M., Hlauschek, C., Kruegel, C., and Kirda, E. (2009). Scalable, behavior-based malware clustering. In *NDSS*, volume 9, pages 8–11. Citeseer.

Bengio, Y. (2009). *Learning deep architectures for AI*. Now Publishers Inc.

Bhatia, T. and Kaushal, R. (2017). Malware detection in android based on dynamic analysis. In *2017 International Conference on Cyber Security And Protection Of Digital Services (Cyber Security)*, pages 1–6. IEEE.

Biggio, B., Corona, I., Maiorca, D., Nelson, B., Šrndić, N., Laskov, P., Giacinto, G., and Roli, F. (2013). Evasion attacks against machine learning at test time. In *Joint European conference on machine learning and knowledge discovery in databases*, pages 387–402. Springer.

Biggio, B., Nelson, B., and Laskov, P. (2012). Poisoning attacks against support vector machines. *arXiv preprint arXiv :1206.6389*.

Borello, J.-M. and Mé, L. (2008). Code obfuscation techniques for metamorphic viruses. *Journal in Computer Virology*, 4(3) :211–220.

Brian Mwandau, M. N. (2020). Image représentant l'anatomie d'un neurone biologique et un neurone artificiel. `https://www.researchgate.net/publication/325870973_Investigating_Keystroke_Dynamics_as_a_Two-Factor_Biometric_Security`. Consulté en août 2021.

Buduma, N. and Locascio, N. (2017). *Fundamentals of deep learning : Designing next-generation machine intelligence algorithms*. " O'Reilly Media, Inc.".

C3.AI (2021). Les techniques de classification et de régression. `https://c3.ai/introduction-what-is-machine-learning/supervised-learning/`. Consulté en août 2021.

CCCS-CIC-AndMal (2020). Andmal 2020 dataset. `https://www.unb.ca/cic/datasets/andmal2020.html`. Consulté en juillet 2021.

Chaulagain, D., Poudel, P., Pathak, P., Roy, S., Caragea, D., Liu, G., and Ou, X. (2020). Hybrid analysis of android apps for security vetting using deep learning. In *2020 IEEE Conference on Communications and Network Security (CNS)*, pages 1–9. IEEE.

Chen, B., Deng, W., and Du, J. (2017). Noisy softmax : Improving the generalization ability of dcnn via postponing the early softmax saturation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5372–5381.

Cheng, C. F. (2020). Exemple de fonctionnement de l'apprentissage par renforcement. `https://www.inwinstack.com/blog-en/blog_ai-en/6262/`. Consulté en août 2021.

Chollet, F. (2017). *Deep learning with Python*. Simon and Schuster.

Datarobot (2021). Illustration des problèmes d'apprentissage. https://www.datarobot.com/wiki/underfitting/. Consulté en août 2021.

Data_Scientest (2020). Machine learning : Définition, fonctionnement, utilisations. https://datascientest.com/machine-learning-tout-savoir. Consulté en juillet 2021.

Data_Scientist_Bial-R (2020). Image représentant la relation entre le machine learning et le deep learning. https://www.bial-r.com/2019/05/22/comprendre-le-machine-learning-et-le-deep-learning/. Consulté en août 2021.

Donges, N. (2018). Recurrent neural networks and lstm. *Towards Data Science*.

Doshi, S. (2019). Various optimization algorithms for training neural network. *Towards Data Science*, 13.

Encyclopaedia_Britannica (2020). Android operating system. https://www.britannica.com/technology/Android-operating-system. Consulté en septembre 2021.

Feng, Y., Anand, S., Dillig, I., and Aiken, A. (2014). Apposcopy : Semantics-based detection of android malware through static analysis. In *Proceedings of the 22nd ACM SIGSOFT international symposium on foundations of software engineering*, pages 576–587.

Fereidooni, H., Conti, M., Yao, D., and Sperduti, A. (2016). Anastasia : Android malware detection using static analysis of applications. In *2016 8th IFIP international conference on new technologies, mobility and security (NTMS)*, pages 1–5. IEEE.

Gabormelli (2021). La fonction heaviside qui ressemble à une marche d'escalier. https://www.gabormelli.com/RKB/Heaviside_Step_Activation_Function. Consulté en août 2021.

Gargenta, M. (2011). *Learning android*. " O'Reilly Media, Inc.".

Géron, A. (2019). *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow : Concepts, Tools, and Techniques to Build Intelligent Systems*. O'Reilly Media.

Giuseppe Ciaburro, B. V. (2021). Exemple de fonctionnement de la convolution. https://www.oreilly.com/library/view/neural-networks-with/9781788397872/70ba8233-7507-474e-9958-4d2d338b5a44.xhtml. Consulté en août 2021.

Google_Play (2021). Google play store. https://play.google.com/store/apps. Consulté en juin 2021.

Hernán, M. A., Hsu, J., and Healy, B. (2019). A second chance to get causal inference right : a classification of data science tasks. *Chance*, 32(1) :42–49.

Jeffares, A. (2018). Exemple de fonctionnement de l'apprentissage non supervisé. https://towardsdatascience.com/supervised-vs-unsupervised-learning-in-2-minutes-72dad148f242. Consulté en août 2021.

Jiang, T., Gradus, J. L., and Rosellini, A. J. (2020). Supervised machine learning : a brief primer. *Behavior Therapy*, 51(5) :675–687.

Jordan, J. (2018). Illustration les différents taux d'apprentissage. https://www.jeremyjordan.me/nn-learning-rate/. Consulté en août 2021.

Karpathy (2015). Différentes architectures des rnns. http://karpathy.github.io/2015/05/21/rnn-effectiveness/. Consulté en août 2021.

Kaspersky (2021). What is a drive by download. https://www.kaspersky.com/resource-center/definitions/drive-by-download. Consulté en août 2021.

Kaspersky_Mobile_Security (2021). Android mobile security threats. https://www.kaspersky.com/resource-center/threats/mobile. Consulté en août 2021.

Kathuria, A. (2018). Intro to optimization in deep learning : Gradient descent. *Hello Paperspace*.

Keyes, D. S., Li, B., Kaur, G., Lashkari, A. H., Gagnon, F., and Massicotte, F. (2021). Entroplyzer : Android malware classification and characterization using entropy analysis of dynamic characteristics. In *2021 Reconciling Data Analytics, Automation, Privacy, and Security : A Big Data Challenge (RDAAPS)*, pages 1–12. IEEE.

Kim, H., Smith, J., and Shin, K. G. (2008). Detecting energy-greedy anomalies and mobile malware variants. In *Proceedings of the 6th international conference on Mobile systems, applications, and services*, pages 239–252.

Kirillov, A. (2018). Réseau de neurone récurrent standard avec fonction d'activation tanh. https://www.codeproject.com/Articles/1272354/ANNT-Recurrent-neural-networks. Consulté en août 2021.

Kremer, S., Mé, L., Rémy, D., and Roca, V. (2019). Cybersécurité.

Kuo, W.-C. and Lin, Y.-P. (2018). Malware detection method based on cnn. In *International Computer Symposium*, pages 608–617. Springer.

Lee, W. Y., Saxe, J., and Harang, R. (2019). Seqdroid : Obfuscated android malware detection using stacked convolutional and recurrent neural networks. In *Deep learning applications for cyber security*, pages 197–210. Springer.

l'Institute AVTest (2021). Malware. https://www.av-test.org/en/statistics/malware/. Consulté en octobre 2021.

Liu, Y., Tantithamthavorn, C., Li, L., and Liu, Y. (2021). Deep learning for android malware defenses : a systematic literature review. *arXiv preprint arXiv :2103.05292*.

Ma, J. (2016). Architecture d'un réseau de neurone récurrent. https://medium.com/@jianqiangma/all-about-recurrent-neural-networks-9e5ae2936f6e. Consulté en août 2021.

Machine_Learning_Mastery (2021). Supervised and unsupervised machine learning algorithms. machinelearningmastery.com/supervised-and-unsupervised-machine-learning-algorithms. Consulté en juillet 2021.

Mahmood, R., Mirzaei, N., and Malek, S. (2014). Evodroid : Segmented evolutionary testing of android apps. In *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pages 599–609.

Malware_Bytes (2021). All about malware. https://www.malwarebytes.com/malware. Consulté en août 2021.

Mishra, A. (2018). Metrics to evaluate your machine learning algorithm. https://towardsdatascience.com/metrics-to-evaluate-your-machine-learning-algorithm-f10ba6e38234. Consulté en octobre 2021.

Mitchell, T. (1997). Machine learning. new york : Mcgraw-hill.

Ml4a (2021). Neural networks. https://ml4a.github.io/ml4a/neural_networks/. Consulté en août 2021.

Mohajon, J. (2020). Confusion matrix for your multi-class machine learning model. https://towardsdatascience.com/confusion-matrix-for-your-multi-class-machine-learning-model-ff9aa3bf7826. Consulté en octobre 2021.

Mohammed, A. R., Viswanath, G. S., Anuradha, T., et al. (2019). Malware detection in executable files using machine learning. In *International Conference on Emerging Trends in Engineering*, pages 277–284. Springer.

National_Center (2021). National center for highperformance computing. https://www.nchc.org.tw/tw/. Consulté en juillet 2021.

Peters, M. E., Neumann, M., Iyyer, M., Gardner, M., Clark, C., Lee, K., and Zettlemoyer, L. (2018). Deep contextualized word representations. *arXiv preprint arXiv :1802.05365*.

Rahali, A., Lashkari, A. H., Kaur, G., Taheri, L., Gagnon, F., and Massicotte, F. (2020). Didroid : Android malware classification and characterization using deep image learning. In *2020 the 10th International Conference on Communication and Network Security*, pages 70–82.

Ramakrishna, M., Satish, A. R., and Krishna, P. S. (2021). Design and development of an efficient malware detection using ml. In *Proceedings of International Conference on Computational Intelligence and Data Engineering*, pages 423–433. Springer.

Rosebrock, A. (2017). *Deep learning for computer vision with Python : starter bundle*. PyImageSearch.

Saha, S. (2018). A comprehensive guide to convolutional neural networks—the eli5 way. *Towards data science*, 15.

Shabtai, A., Moskovitch, R., Elovici, Y., and Glezer, C. (2009). Detection of malicious code by applying machine learning classifiers on static features : A state-of-the-art survey. *information security technical report*, 14(1) :16–29.

Shaikh, F. (2017). Deep learning vs. *Machine Learning—The Essential Differences you Need to Know*.

Sourav, S., Khulbe, D., and Kapoor, N. (2019). Deep learning based android malware detection framework. *arXiv preprint arXiv :1912.12122*.

Srihari, S. N. (2021). Data augmentation. https://cedar.buffalo.edu/~srihari/CSE676/7.4%20DataAugmentation.pdf. Consulté en août 2021.

TechTalks (2021). Minimum local et minimum global. https://bdtechtalks.com/2020/04/27/deep-learning-mode-connectivity-adversarial-attacks/gradient-descent-local-minima/. Consulté en août 2021.

Vidas, T., Tan, J., Nahata, J., Tan, C. L., Christin, N., and Tague, P. (2014). A5 : Automated analysis of adversarial android applications. In *Proceedings of the 4th ACM Workshop on Security and Privacy in Smartphones & Mobile Devices*, pages 39–50.

Virus_Share (2021). Virus share. https://virusshare.com/. Consulté en juin 2021.

Wang, Z., Cai, J., Cheng, S., and Li, W. (2016). Droiddeeplearner : Identifying android malware using deep learning. In *2016 IEEE 37th Sarnoff Symposium*, pages 160–165. IEEE.

Wei, F., Li, Y., Roy, S., Ou, X., and Zhou, W. (2017). Deep ground truth analysis of current android malware. In *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*, pages 252–276. Springer.

Wong, M. Y. and Lie, D. (2016). Intellidroid : A targeted input generator for the dynamic analysis of android malware. In *NDSS*, volume 16, pages 21–24.

Yan, S. (2016). Understanding lstm and its diagrams. https://blog.mlreview.com/understanding-lstm-and-its-diagrams-37e2f46f1714. Consulté en juillet 2021.

Yang, F., Zhuang, Y., and Wang, J. (2017). Android malware detection using hybrid analysis and machine learning technique. In *International Conference on Cloud Computing and Security*, pages 565–575. Springer.

Yerima, S. (2018). Android malware dataset for machine learning. https://figshare.com/articles/dataset/Android_malware_dataset_for_machine_learning_2/5854653. Consulté en juillet 2021.

Yuan, Z., Lu, Y., Wang, Z., and Xue, Y. (2014). Droid-sec : deep learning in android malware detection. In *Proceedings of the 2014 ACM conference on SIGCOMM*, pages 371–372.

Yuan, Z., Lu, Y., and Xue, Y. (2016). Droiddetector : android malware characterization and detection using deep learning. *Tsinghua Science and Technology*, 21(1) :114–123.

Zefferer, T., Teufl, P., Derler, D., Potzmader, K., Oprisnik, A., Gasparitz, H., and Hoeller, A. (2013). Power consumption-based application classification and malware detection on android using machine-learning techniques. *Future Computing*, 5 :26–31.

Zhou, Y. and Jiang, X. (2012). Dissecting android malware : Characterization and evolution. In *2012 IEEE symposium on security and privacy*, pages 95–109. IEEE.