République Algérienne Démocratique et Populaire
Ministère de l'enseignement supérieur et de la recherche scientifique
Université Larbi Tébessi - Tébessa
Faculté des Sciences Exactes et des Sciences de la Nature et de la Vie
Département : Mathématiques et Informatique

Mémoire de fin d'études
Pour l'obtention du diplôme de MASTER
Domaine : Mathématiques et Informatique
Filière : Informatique
Option : Réseaux et sécurité informatique

Thème

Système de détection de malwares basé sur l'apprentissage profond pour Android

Présenté Par :
Maini Rachid Cherif

Devant le jury :

Dr Bendib Issam	*MCB*	*Univérsité Larbi Tébessi*	*Président*
Dr Gasmi Mohamed	*MCB*	*Univérsité Larbi Tébessi*	*Examinateur*
Pr Derdour Makhlouf	*Prof*	*Univérsité Larbi Tébessi*	*Encadreur*

Date de soutenance : *28/06/2020*


بسم الله الرحمن الرحيم

REMERCIEMENTS

Je voudrais tout d'abord exprimer mes plus profonds remerciements à mon encadreur ***Pr.Derdour Makhlouf*** *pour son accord d'être mon directeur de mémoire et de sa disponibilité et son aide pendant toute la préparation de ce travail.*

Je tiens aussi à remercier tous les membres du jury : ***Dr. Bendhib Issam*** *et* ***Dr.Gasmi Mohamed****, pour leur disponibilité et acceptation d'examiner et de rapporter mon travail.*

Mes plus profonds remerciements à ***Dr. Ahmim ahmed*** *pour ses conseils et sa disponibilité et son aide pendant toute la préparation de ce travail.*

Je remercie ainsi tous les enseignements du département Mathématiques et informatique.

Je ne saurais oublier de remercier ma mère ainsi que ma sœur et toute ma famille pour leurs soutien moral, leurs encouragements et leurs patience durant les étapes de réalisation de ce travail.

Enfin, Que tous ceux qui directement ou indirectement m'ont apporté leurs aide, trouvent ici l'expression de mes sincères remerciements.

DÉDICACES

Je dédie ce modeste travail spécialement
A ma chére mère et sœur et à toute ma famille.
A la mémoire de mon très cher père
symbole de courage, de tendresse et
à mon meilleur ami Amar Nahal,
que Dieu le puissant leur accorde sa
clémence et les accueille en son vaste paradis.
A tous les enseignants et les collègues d'études.

ملخص

أصبح نظام التشغيل أندرويد أكثر أنظمة التشغيل المحمولة استخداماً، وقد جذبت هذه الشعبية المتزايدة انتباه مطور البرامج الضارة، مما يمنحهم إمكانية إنشاء تطبيقات ضارة لمهاجمة أجهزة أندرويد. في الآونة الأخيرة، هناك جيل جديد من برامج أندرويد الضارة التي تجعل من الصعب اكتشافها باستخدام الطرق التقليدية. نقترح في هذه الأطروحة طريقة توصيف واكتشاف هذا البرنامج الضار بناءً على التفويضات ومكالمات واجهة برمجة تطبيق والأوامر ذات الصلة التي تستغل وضع هذه العدوى. تعتمد طريقتنا على تقنيات التعلم العميق، والتي تسمح لنا بإنشاء أفضل نموذج فيما يتعلق بمجموعة البيانات المستخدمة. تم اختبار نموذجنا على أجهزة الأندرويد الحقيقية لتحديد ما إذا كان تطبيق معين ضارًا أم لا من خلال إجراء تحليل للسلوك.

الكلمات الرئيسية: أندرويد ، التحليل الثابت والديناميكي، التعلم العميق، اكتشاف الأضرار ، التفويضات، مكالمات واجهة برمجة تطبيق، الأوامر ذات الصلة، الشبكات العصبية المتكررة.

ABSTRACT

The Android operating system has become the most used mobile operating system, this growing popularity has caught the attention of the malware developer, which gives them the possibility of creating malicious applications to attack Android devices. Recently, there is a new generation of Android malware that makes it much more difficult to detect using conventional methods. We propose in this thesis a method of characterization and detection of this malware based on permissions, API calls and related commands which exploit this mode of infection. Our method is based on deep learning techniques, which allows us to create the best model in relation to the data set used. Our model has been tested on real Android devices to determine whether a particular application is malicious or not by performing a behavior analysis.

Key words : Android, static and dynamic analysis, deep learning, malware detection,permissions, API calls, related commands, recurrent neural networks.

RÉSUMÉ

Le système d'exploitation Android est devenu le système d'exploitation mobile le plus utilisé, cette popularité croissante a attiré l'attention du développeur des malwares, ce qui leur donne la possibilité de crée des applications malveillantes pour attaquer les appareils Android. Récemment, il existe une nouvelle génération de logiciels Android malveillants qui les rendent beaucoup plus difficiles à détecter a l'aide de méthodes conventionnelles. Nous proposons dans ce mémoire une méthode de caractérisation et de détection de ces malwares basés sur les permissions, appels d'API et les commandes liées qui exploitent ce mode d'infection. Notre méthode est basée sur les techniques d'apprentissage profond, ce qui nous a permet de crée le meilleur modèle par rapport à l'ensemble de données utilisé. Notre modèle a été testé sur de vrais appareils Android pour déterminer si une application particulière est malveillante ou non en effectuant une analyse de comportement.

Mots clés : Android, analyse statique et dynamique, apprentissage profond, détection des malwares, permissions, appels API, commandes liées, réseaux de neurones récurrents.

TABLE DES MATIÈRES

LISTE DES ABRÉVIATIONS

SE Système d'exploitation.

OS Operating System.

IPC Communication Inter-Processus

XNU X is Not Unix.

API Application Programming Interface.

XML Extensible Markup Language.

APK Android Package Kit.

GID Group ID.

UID User ID.

DAC Discretionary Access Control.

IDS Intrusion detection System.

ML Machine Learning.

DL Deep Learning.

AI Artificial Intelligence.

SSL Semi-Supervised Learning.

PNL Natural Language Processing.

SDK Software Development Kit.

CSV Comma-separated values.

JSON JavaScript Object Notation.

PCA Principal Component Analysis.

KDD knowledge discovery in databases.

TABLE DES FIGURES

LISTE DES TABLEAUX

INTRODUCTION GÉNÉRALE

De nos jours, les ordinateurs ont été remplacés par des appareils plus portables comme des bracelets, des appareils mobiles intelligents, des tablettes, etc. avec l'avancement de la technologie. Le système d'exploitation Android (OS) est l'un des systèmes d'exploitation les plus populaires utilisés sur ces appareils. Il existe des millions d'applications sur le système d'exploitation Android, et les utilisateurs peuvent facilement télécharger leurs applications sur leurs appareils mobiles via le magasin des applications Android. Bien que les appareils mobiles facilitent la vie, les développeurs de logiciels malveillants tentent d'accéder aux informations personnelles via ces applications qui facilitent la vie. Ils peuvent accéder aux appareils des utilisateurs en injectant des logiciels malveillants dans un fichier apk qui représente une extension des applications basées sur Android.

Les premiers travaux d'Android sont concentré sur l'analyse des limites de sa sécurité et la manière de les couvrir. Ce type d'approche a cependant une principale limite à savoir ne pas pouvoir détecter et apprendre de nouvelles attaques. D'autres travaux sont basés sur les flux d'informations[14], où il faut apprendre comment les attaques ont lieu en analysant directement les applications malveillantes et en utilisant la base de connaissance acquise durant l'apprentissage pour détecter ces malwares.

Plusieurs travaux ont été développés dans le domaine de l'apprentissage automatique pour la détection des malwares android en utilisant plusieurs algorithmes pour l'extraction des caractéristiques (statistiques ou structurelles) et des classificateurs. Ces travaux ont prouvé leurs puissances en termes du taux de détection sur un ensemble de données qui contient un nombre de caractéristiques bas. Tout de même, ces résultats restent limités dans le cadre de traitement de très grand nombre de caractéristiques.

L'apprentissage profond est apparu spécialement pour résoudre les problèmes rencontrés de l'apprentissage automatique, l'un des ingrédients les plus importants pour le succès de ces méthodes est la disponibilité de grandes quantités de données d'entraînement et la diversité des caractéristiques.

Dans ce contexte et dans le cadre de notre projet de fin d'étude, nous avons organisé ce manuscrit de la façon suivante :

Une introduction générale,
Le premier chapitre est consacré au contexte d'étude, Nous allons présenter les Applications mobiles en mettant l'accent sur ces systèmes d'exploitations mobile les plus répandu actuellement. Ensuite nous détaillons le système d'exploitation Android en illustrant son architecture, ainsi que les stratégies de développement d'applications Android. Enfin nous allons présenter la sécurité du système Android.
Dans **le deuxième chapitre**, nous allons présenter les limites de mécanismes de sécurité d'Android. Ensuite nous aborderons les logiciels malveillants en commençant par la définition, le cycle de vie, la famille et les techniques de détection de ce dernier. Enfin nous conclurons avec les contre-mesures qui sont appliquées pour réduire les risques d'attaques contre ce système.
Le troisième chapitre, dans ce chapitre nous allons définir tous d'abord l'apprentissage automatique,ces différents types, ainsi que les modèles les plus utilisé. Ensuite nous allons introduire l'apprentissage profond en commençant par ces définitions, son histoire, ces différents modèles, ces métriques d'évaluation ainsi que ces fonctions d'activation. Enfin nous allons présenter les principaux problèmes de l'apprentissage profond et les différentes méthodes pour les éviter.
Les travaux connexes et les approches qui ont été réalisé en utilisant l'apprentissage automatique et profond sont présenté dans **le quatrième chapitre**.
Le cinquième chapitre, représente notre contribution, dans lequel nous détaillons notre architecture pour la détection des malwares Android, ainsi que l'implémentation, où nous présenterons l'ensemble de données utilisé, les résultats expérimentaux et les outils fournis pour l'implémentation de notre contribution proposée suivi par les interfaces de notre application.

Enfin, Nous conclurons ce manuscrit en présentant quelques perspectives ouvertes par notre travail.

CHAPITRE 1

APPLICATIONS MOBILES ET SÉCURITÉ

1.1 Introduction

Sans aucun doute, les logiciels malveillants ne cessent d'augmenter pour les applications mobiles. En conséquence, les chercheurs dépensent des ressources importantes pour améliorer les techniques de détection des logiciels malveillants. Afin de comprendre le comportement de ces malicieux logiciels et analyser ceux d'Android, il est nécessaire de s'intéresser au fonctionnement du SE lui-même, et aux applications Android. Dans ce premier chapitre, nous présentons tout d'abord les applications mobiles en mettant l'accent sur les systèmes d'exploitation mobiles les plus répandu actuellement. Ensuite, nous enchaînons dans le système d'exploitation Android en précisant son architecture, ainsi que les stratégies de développement des applications Android. Enfin nous présentons la sécurité de ce système d'exploitation mobile.

1.2 Les applications mobiles

Les applications mobiles sont constituées de logiciels / ensembles de programmes qui s'exécutent sur un appareil mobile et effectuent certaines tâches pour l'utilisateur. L'application mobile peut être utilisée dans la plupart des appareils mobiles, y compris les appareils mobiles peu coûteux et de base. L'application mobile a de multiples utilisations pour son vaste domaine de fonctionnement, comme appeler, envoyer des messages, naviguer, discuter, communiquer sur les réseaux sociaux, audio, vidéo, jeux, etc [30].

1.2.1 Systèmes d'exploitation mobiles

Tout comme un ordinateur dispose d'un système d'exploitation, les appareils mobiles aussi. Actuellement connu sous le nom d'OS (Operating System) mobile.

Un système d'exploitation (OS) est un ensemble de logiciels qui gère les ressources matérielles informatiques et fournit des services communs pour programmes informatiques. Le système d'exploitation est un composant essentiel du logiciel système dans un système informatique. Les programmes d'applications nécessitent généralement un système d'exploitation pour fonctionner [28].

IOS

iOS est un SE mobile développé par Apple. Il s'appelait à l'origine iPhone OS, mais a été renommé iOS en juin 2009. L'iOS fonctionne actuellement sur iPhone , iPod touch et iPad, dont ils partagent les fondations : un noyau XNU basé sur le micro-noyau Mach, les différents services Unix et Cocoa, etc. iOS possède 4 couches d'abstractions similaires à macOS [37] :

— Une couche « noyau OS ».

— Une couche « noyau Services ».

— Une couche « Media ».

— Une couche « Cocoa ».

FIGURE 1.1 – Ios (operating System) [37].

Android

Lancé en 2005 par la start-up du même nom et ensuite racheté par Google en 2007, Android est un SE mobile fondé sur un noyau Linux. Il est considéré comme une « pile de logiciels » se comportant comme un système d'exploitation. Cette pile comporte les 3 couches suivantes [37] :

— Un SE comportant le noyau Linux et ses pilotes.

— Les applications.

— Les logiciels qui sont entre le système et les applications.

FIGURE 1.2 – Android (OS) [37].

1.2.2 Système Android

Architecture du système Android

Le système d'exploitation Android est une pile de composants logiciels qui peuvent être divisés en quatre couches : 1) Couche de noyau. 2) Couche des bibliothèques natives. 3) Couche Framework d'application. 4) Couche d'application. Le schéma d'architecture du système d'exploitation Android est illustré dans la figure 1.3. Chaque couche exécute un ensemble spécifique de tâches et communique aux autres couches via des interfaces clairement définies. [25].

a) **Couche de noyau :**

La couche de noyau fournit des fonctionnalités de base du système telles que la gestion des processus, la gestion de la mémoire, la gestion des appareils, y compris la caméra, l'affichage, le clavier, etc. La raison du choix du noyau Linux pour Android OS est que Linux est vraiment bon pour les opérations de base telles que la mise en réseau, une vaste gamme de pilotes de périphériques qui éliminent la douleur de l'interfaçage avec le matériel périphérique. Pour certains besoins particuliers, le noyau amélioré de Google permet de mieux répondre aux besoins des plates-formes mobiles telles que la gestion de la mémoire, la gestion des processus, la gestion de l'alimentation, le mécanisme IPC (environnement interprocessus spécial de comminucation) et une meilleure gestion des ressources système limitées[25].

b) **Couche des bibliothèques natives :**

Au sommet de la couche du noyau Linux se trouvent les bibliothèques natives d'Android. Cette couche permet à l'appareil de gérer différents types de données qui sont spécifiques au matériel, comme le montre la figure 1.3. Elle possède deux parties essentielles, l'une est la bibliothèque Android et la seconde est le runtime Android. Toutes ces bibliothèques sont écrites en langage de programmation C++. Elles effectuent la plupart des tâches de levage lourdes et fournissent beaucoup de puissance derrière la plate-forme Android. Ces bibliothèques sont appelées via l'interface java [25].

c) **Couche Framework d'application :**

La couche de structure d'application est au-dessus de la couche de bibliothèque native. La couche application fournit une interface de programmation d'application (API) majeure et des services de niveau supérieur sous forme de classes java. Les développeurs d'applications sont autorisés a accéder à tous les structures d'API pour les programmes principaux qui simplifient la réutilisation des ces composants. Ces API sont ouvertes à tous pour créer des applications Android. Il existe différents types de composants d'application. Chaque type a un cycle de vie et un objectif différent qui décrit comment le composant sera créé et détruit[25].

d) **Couche d'application :**

Dans l'architecture d'Android, la couche d'application est la couche supérieure. Ceux-ci comprennent à la fois l'application native à pré-installée avec chaque appareil, comme : numéroteur téléphonique, client SMS, gestionnaire de contacts du navigateur Web, etc. Un utilisateur moyen de l'appareil Android interagirait principalement avec cette couche pour des fonctions de base comme passer des appels téléphoniques, l'envoi de messages texte, la capture d'images, la navigation sur le Web, la lecture de vidéos et d'audios. Un développeur peut câbler sa propre application et la remplacer par l'application existante. Il existe également de nombreuses applications tierces disponibles dans Google Play Store. L'utilisateur peut facilement télécharger et utiliser ces applications en fonction de leurs besoins[25].

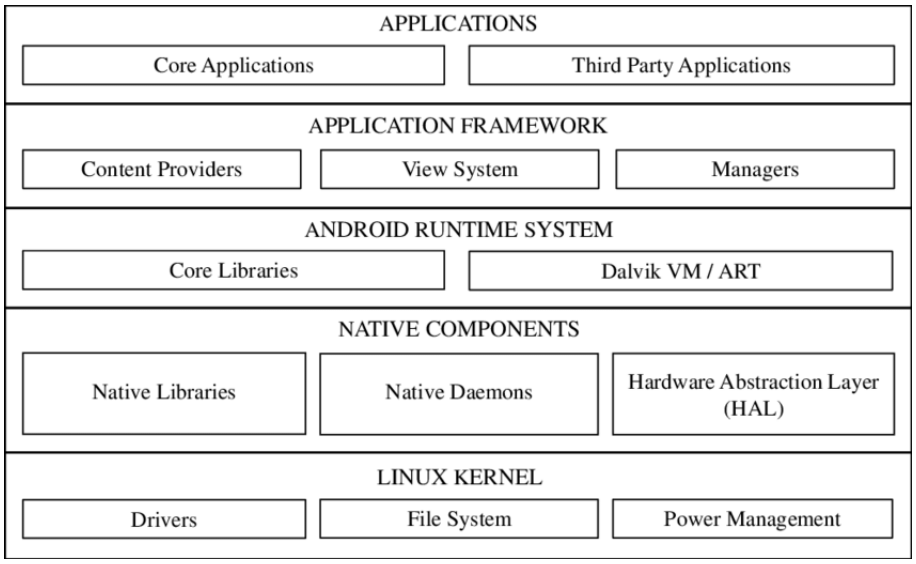


FIGURE 1.3 – L'architecture du SE Android [25].

1.2.3 Stratégies de développement

La conception d'applications mobiles se produit suivant trois stratégies de développement différentes. Dans ce qui suit, Nous présenterons un bref aperçu de chaque stratégie[45].

Application native

Une application native est développée spécifiquement pour une seule plateforme, grâce aux outils conçus pour celle-ci. Pour développer une application pour deux plateformes différentes, il vaut mieux de développer deux applications distinctes[45].

Application Web

Au contraire à une application native, une application web est une application mobile développée avec les outils de développement web actuel : HTML5, CSS3 et JavaScript. C'est une application qui une fois développée est accessible et exécutable sur tous les appareils mobiles via leur navigateur web[45].

Application hybride

Une application hybride est une application qui combine les éléments d'une application web et native. Elle repose essentiellement sur la solution Cordova/PhoneGap, cette solution sert de passage entre le langage web et le natif et elle nous permet d'utiliser un unique outil pour le développement Web pour tous les SE mobiles (IOS, Android).[45].

1.2.4 Les applications Android

Une application Android est spécifiquement développée pour les appareils mobiles et les tablettes utilisant le système Android. Elles sont de nature très variables tel que les applications jeux, mobile commerce, utilitaire, service d'informations [17]. Pour mieux comprendre le fonctionnement d'une application android nous présenterons son architecture, communication entre les composants (Intents), Intent-filter et les package Android, ainsi que AndroidManifest.xml.

Architecture d'une application Android : les différents composants

Une application Android peut contenir plusieurs composants, chacune d'elles peut-être un point d'entrée dans le programme.

Il existe quatre types de composants : activity, service et ContentProvider, et BroadcastReceiver. Ces pièces portent des informations à travers des messages appelés Intents[14].

a) **Activity :** L'activité est l'un des éléments constitutifs d'Android OS. En terme simple, l'activité est un écran avec lequel l'utilisateur interagit. Chaque activité dans Android a un cycle de vie comme créé, démarré, repris, mis en pause, arrêté ou détruit. Ces différents états sont connus sous le nom de Cycle de vie d'activité. En d'autres termes, nous pouvons dire que l'activité est une classe pré-écrite en programmation Java [14].

b) **ContentProvider :** Est un composant servant au partage de données d'une application, qui sert d'interface entre l'application souhaitant accéder aux données. Elles sont stockées dans

une base de données locale SQLite mais aucune restriction n'est imposée sur la manière de stocker ces données. Par exemple les contacts sont stockés dans une base de données locale dont l'accès se fait à travers un composant de type ContentProvider[14].

c) **Service :** Est un composant effectuant des tâches en arrière-plan. Il est utilisé pour exécuter de longues tâches internes ou a exécuter une tâche à la demande d'une application [14].

d) **BroadcastReceiver :** Est un composant utilisé pour écouter les messages en large diffusion sur le système. Lorsqu'un nouveau SMS est reçu par l'appareil mobile, le système envoie un message en diffusion pour notifier les différentes applications d'envoi et réception de SMS.[14].

Intents : communication entre composants

Un Intent est un message avec une demande d'action et optionnellement, des données réalisées à l'intention d'une autre composante utilisant un Intent Filter indiquant qu'elle accepte ce type de message. Les trois principales utilisations sont de lancer une activité, un Service ou d'effectuer une diffusion [14].

Un Intent est explicite s'il déclare la composante destinatrice explicitement et dit implicite s'il laisse les paramètres du système et/ou l'utilisateur détermine la composante (ou l'application) qui traite ce message[14].

Intent-Filter

Description structurée des valeurs d'intention à faire correspondre. Un Intent-Filter peut apparier des actions, des catégories et des données (via son type, son schéma et / ou son chemin) dans un Intent. Il inclut également une valeur "priorité" utilisée pour ordonner plusieurs filtres correspondants [2].

Les objets Intent-Filter sont souvent crées en XML dans le fichier manifeste d'Android, à l'aide des balises d'intent-filter nous pouvant filtrer trois caractéristiques d'intention : l'action, les données et les catégories [2].

```
<activity android:name=".MainActivity">
    <!-- This activity is the main entry, should appear in app launcher -->
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
        <data android:mimeType="text/plain"/>
    </intent-filter>
</activity>
<activity android:name=".ResultActivity">
    <!-- This activity handles "SEND" actions with text data -->
    <intent-filter>
        <action android:name="android.intent.action.SEND"/>
        <category android:name="android.intent.category.DEFAULT"/>
        <data android:mimeType="text/plain"/>
    </intent-filter>
</activity>
```

FIGURE 1.4 – Example Intent-Filter dans Manifest android [[2].

Android Package

Les objets contiennent des informations sur la version concernant la mise en œuvre et la spécification d'un package Java. Ces informations de version sont récupérées et rendues disponibles par la classe **Loaderinstance**. En règle générale, il est stocké dans le manifeste distribué avec les classes [3].

L'ensemble de classes qui compose le package peut implémenter une spécification particulière et, le cas échéant, le titre de la spécification, le numéro de version et les chaînes du fournisseur identifient cette spécification. Une application peut demander si le package est compatible avec une version particulière[3].

AndroidManifest.xml

Le fichier **AndroidManifest.xml** décrit les informations essentielles dans l'application (fichier APK), certaines parties importantes qui peuvent être mentionnées sont [43] :

- **Nom du package :** L'identifiant unique d'une application dans Google Play Store et l'appareil.
- **Composants d'application :** Les activités, les services, les fournisseurs de contenu et les récepteurs de diffusion définissent les comportements des applications et leurs capacités d'interagir avec le système d'exploitation.
- **Permissions manifestes :** décrivez ce dont l'application a besoin pour accéder aux informations privilégiées du système ou d'autres applications, et déclare également ce dont les autres applications ont besoin pour accéder à cette application. Selon la documentation

Android, le niveau de protection caractérise le risque potentiel impliqué dans chaque permission de fabricant et permet au système de donner suite aux permissions accordées sur l'application.

La figure 1.5 présente un example du fichier **AndroidManifest.xml** d'une application sous Android Studio [1].

AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.mrcherif.nmap_auto">

    <uses-permission android:name="android.permission.INTERNET" />
    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        <activity android:name=".ThreadMain">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <activity android:name=".MainActivity" />
        <activity android:name=".hidtroy" />
    </application>

</manifest>
```

FIGURE 1.5 – Fichier AndroidManifest.xml.

1.3 Sécurité du système Android

Android implémente nativement quelques mécanismes qui offrent un certain niveau de sécurité.

1.3.1 Permissions

Le but d'une permission est de protéger la confidentialité d'un utilisateur Android. Les applications Android doivent avoir la permission pour accéder aux données utilisateur sensibles (telles que les contacts et les appels), ainsi qu'à certaines fonctionnalités du système (telles que l'appareil photo et Internet). Selon la fonctionnalité, le système peut accorder automatiquement l'autorisation ou peut inviter l'utilisateur a approuver la demande[35].

1. Environnement de développement pour développer des applications mobiles Android.

Chaque permission correspond à un GID [2] du noyau Linux. Chaque GID possède les accès aux ressources du SE requises pour l'exécution des comportements rattachés à cette permission.

Pour chaque permission accordée, Android ajoute un UID de l'application au groupe qui y correspond. l'application obtient à ce moment les privilèges qui lui permettent d'agir avec la permission demandée[35]. Le tableau 1.1 résume les quatre types de permissions.

Type	Description
Normal	Valeur par défaut des permissions. Elle est automatiquement accordée à toute application la demandant.
dangereuse	L'utilisateur doit accepter la permission afin de la lui accorder. Exemple read-sms pour l'accès aux SMS.
signature	Permission accordée uniquement si l'application demandeuse est signée avec le certificat du développeur ayant déclarée la permission.
signatureOrSystem	Permission accordé seulement aux applications du systéme, plus précisément celles dans la partition system, ou à celles qui ont été signées avec la même signature que l'application qui a déclarée la permission.

TABLE 1.1 – Types de permission Android [14].

1.3.2 Identifiant unique d'utilisateur (UID)

Le contrôle des entrées sur la base des rôles est introduite sous la forme d'identifiant d'utilisateur (User Id, UID). En attribuant un UID par application Android au moment de l'installation et en les obligeant a s'exécuter uniquement via ce UID, chaque application est stocké dans un espace de fichiers séparé des autres applications. La seule exception est s'il s'agit d'un utilisateur privilégié (c.-à-d. root) qui peut outrepasser ces restrictions [35].

1.3.3 Le contrôle d'accès discrétionnaire et le sandboxing

Le mécanisme DAC (Discretionary Access Control) permet le contrôle d'accès des utilisateurs aux fichiers et aux répertoires. Il fonctionne d'une manière invisible pour les développeurs d'applications et les utilisateurs. Il permet de séparer les applications des ressources systèmes. En effet,

2. Groupe ID : est utilisé pour gérer plusieurs utilisateurs dans un système Linux habituel

il est utilisé pour autoriser ou non les applications a accéder aux ressources système, par exemple la possibilité d'activer le Bluetooth ou des sockets réseau et la capacité d'accéder au système de fichiers sur la carte mémoire SD [34].

1.3.4 Administration de l'appareil Android

Android propose une API qui permet de développer des applications afin d'administrer les appareils mobiles. L'API permet d'accroître la politique de sécurité sur les mots de passe (ex : taille, expiration et nombre de fois a ré-entrer le mot de passe), imposer le chiffrement des partitions(activer / désactiver le WI-FI, le bluetooth, etc.), demander la création d'un nouveau mot de passe, verrouiller l'appareil mobile, etc.[14].

1.4 Conclusion

Nous avons présenté dans ce chapitre un aperçu des applications mobiles en général, et particulièrement le système Android, où nous avons expliqué le fonctionnement de cette plateforme ainsi que son architecture. Nous nous sommes intéressés par la suite au mécanismes de sécurité proposé par ce système qui offrent un certain niveau de sécurité.

CHAPITRE 2

MENACES DE SÉCURITÉ ET MALWARES ANDROID

2.1 Introduction

La sécurité du système Android est devenue une préoccupation de plus en plus importante de l'informatique liée à la téléphonie mobile. Elle est particulièrement préoccupante car elle concerne la sécurité de toutes les informations disponibles au sein des appareils mobiles. Dans ce chapitre nous présentons les limites de mécanismes de sécurité Android. Ensuite nous aborderons les logiciels malveillants en commençant par la définition, le cycle de vie, la famille et les techniques de détection de ce dernier. Enfin nous conclurons avec les contre-mesures qui sont appliquées pour réduire les risques d'attaques contre ce système.

2.2 Limites des mécanismes de sécurité Android

2.2.1 Abus de permission :

Les permissions donnent accès aux ressources sensibles de l'appareil mobile aux applications. Si l'utilisateur souhaite installer une application, il doit lui accordé toutes les permissions qu'elle a demandée. Si elles filtrent l'accès a ces ressources, il n'y a aucune vérification au niveau de l'usage de ces ressources. Seul les développeurs des applications permet de s'assurer qu'il n'y aura aucun abus. Les simples attaques utilisent aussi les permissions de manière incorrecte et c'est le cas des malwares qui ont pour but de faire fuir des données sensibles de l'appareil mobile [14].

2.2.2 Permissions : attaques par délégation et attaques par collusion

Une attaque par délégation consiste a déléguer l'exécution de la tâche nécessitante une permission que l'application malveillante n'a pas à une autre application qu'elle la. Par exemple, une application qui n'a pas la permission de communiquer sur le réseau pourrait se servir du navigateur pour entrer des informations ou télécharger des fichiers.

Une attaque par collusion est une coopération entre plusieurs applications pour but de mener une attaque. Il n'existe aucun logiciel malveillant utilisant ce type d'attaque à notre connaissance[14].

2.2.3 Failles logicielles : élévation de privilège

Le système Android a également des failles logicielles, comme tout les systèmes disponibles actuellement. Exploiter certaines d'entre elles permet d'augmenter les privilèges d'une application et ainsi exécuter des opérations sensibles liées aux informations personnelles. [14].

2.3 Les logiciels malveillants Android

Sous sa forme différente, les applications malveillantes représentent un problème majeur affectant le système d'exploitation Android, Cette section décrit le cycle de vie des malwares, sa famille et ces techniques de détection.

2.3.1 Définition

Un malware, abréviation de logiciel malveillant, est un programme ou code dans le principale but est de nuire un système donné. Dans le reste des chapitres nous ferons usage du terme échantillon de malware. Un échantillon d'un malware est une application Android qui contient ce malware. Analyser un malware consiste aussi à analyser un ou plusieurs de ses échantillons pour extraire les informations liées a ce malware, détecter un malware revient a décider si une application donnée est un échantillon d'un malware [14].

2.3.2 Cycle de vie des logiciels malveillants

Les logiciels malveillants pour les plates-formes mobiles en général et Android en particulier reproduisent le comportement des virus rencontrés sur les ordinateurs de bureau. Leur cycle de vie est structuré autour de sept phases principales [48]. La figure 2.1 démontre le cycle de vie de malware.

a) **Création :** Étape au cours de laquelle le programmeur conçoit et implémente tout le code malveillant qui sera inclus dans le logiciel malveillant.

b) **Gestation :** Étape au cours de laquelle l'application malveillante s'infiltre et s'installe dans le système qu'elle souhaite infecter. Il reste inactif tout au long de cette étape. C'est pourquoi sa présence reste totalement inconnue pour l'utilisateur.

c) **Reproduction ou infection :** Le malware se reproduit un nombre important de fois avant de se manifester dans cette phase. L'auteur du malware cherche a contrôler à distance les appareils et a accéder aux données privées. Les logiciels malveillants se propagent via le partage de fichiers ou des techniques d'ingénierie sociale sur Android. Il utilise SMS, Bluetooth, Wifi comme moyen de communication et se déguise souvent en application normale.

d) **Activation :** Certains logiciels malveillants activent leur routine de destruction lorsque certaines conditions sont remplies (le compte à rebours interne atteint par exemple). L'activation peut également être effectuée à distance. Le but de cette phase est de s'approprier progressivement toutes les ressources des appareils.

e) **Découverte :** L'utilisateur remarque un comportement étrange et soupçonne la présence d'une application malveillante. Ce comportement étrange peut inclure des pertes de performances, des modifications en cours dans la page d'accueil du navigateur Web ou l'indisponibilité de certaines fonctions du système.

Les antivirus aident souvent l'utilisateur a détecter les actions malveillantes en envoyant des alertes au propriétaire de l'appareil.

f) **Assimilation :** Les antivirus mettent à jour leur base de données virales après la découverte de nouveaux logiciels malveillants. Si possible, une solution ou un antidote est également proposé pour éliminer cette menace.

g) **Élimination :** Il s'agit de la phase où l'antivirus découvrant le logiciel malveillant, invite l'utilisateur à le supprimer. Il marque la mort du malware.

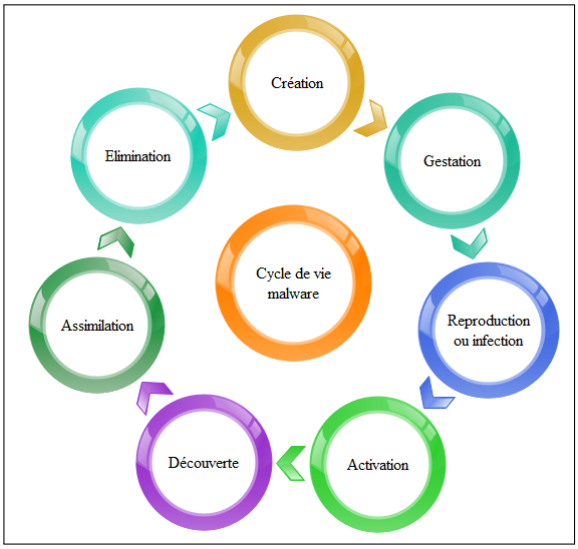


FIGURE 2.1 – Cycle de vie des logiciels malveillants.

2.3.3 Famille de malwares

Il existe des millions de malwares différents. Ces malwares ont des fonctionnalités différentes, ils peuvent être classés par famille [46] :

a) **Backdours :** Qui permet l'exécution d'opérations contrôlées à distance qui endommagent l'appareil.

b) **Spyware commercial :** Qui envoie des informations sensibles sans connaissance des utilisateurs tels que les informations de suivi.

c) **Collecte de données :** Qui extrait les informations sur les applications installées, les comptes d'utilisateurs ou les fichiers de l'appareil sans connaissance de l'utilisateur.

d) **dDownloader hostile :** Qui télécharge d'autres nuisibles applications, bien qu'il n'inclut aucun code.

e) **Fraude par SMS :** Qui fournit des interfaces qui ressemblent à sources fiables. Il utilise ces interfaces pour demander des informations d'authentification ou informations de facturation qui permet à l'utilisateur de les envoyer à un tiers.

f) **Ransomware :** Qui contrôle totalement ou partiellement les mobiles ou les données du mobile en verrouillant l'appareil ou chiffrement des données afin de demander la rançon pour supprimer le contrôle.

g) **Spam :** Qui transmet des messages commerciaux indésirables aux contacts de l'utilisateur.

h) **Spyware :** Qui vole les contacts, les images, les fichiers, le contenu des e-mails, journaux d'appels, journaux de messages et historique du navigateur. En outre, l'enregistrement d'appels téléphoniques ou audio est également envisagé en tant que logiciel espion.

i) **Trojan :** Ce type d'application apparaît comme une application bénigne car il cache ses actions néfastes contre l'utilisateur.

2.3.4 Techniques de détection de malware

Diverses techniques sont concentré sur la détection des malwares Android. Nous allons présenter ci-dessus les techniques de détection de ce dernier et ces outils.

Analyse statique :

L'analyse statique filtre des parties de l'application sans vraiment les exécuter. Cette technique intègre une analyse basée sur les signatures, les autorisations et les composants. La stratégie basée sur la signature dessine des fonctionnalités et crée des signes distinctifs pour identifier des logiciels malveillants spécifiques. Par conséquent, il ne suffit pas de reconnaître la variation ou le logiciel malveillant non identifié. La stratégie basée sur les autorisations reconnaît les demandes d'autorisation pour distinguer les logiciels malveillants. Les techniques basées sur les composants décompilent l'application pour dessiner et inspecter la définition et les connexions de code d'octet

des composants importants (c'est-à-dire les activités, les services, etc.) pour identifier les expositions. Les principaux inconvénients de l'analyse statique sont l'absence de chemins d'exécution réels et de conditions d'exécution appropriées. De plus, il existe des problèmes dans l'occurrence de l'obscurcissement du code et du chargement dynamique du code.[41]

Outils d'analyse statique :

Parmi les outils l'analyse statique d'applications sous Android [18] :

— **Androguard :** Est un framework qui analyse les applications Android.

— **IDA pro version 6.1 :** Désassembleur, un logiciel utilisé pour traduire le code machine dans un format lisible.

— **APKInspector :** Outil en interface graphique permettant d'analyser une application Android.

— **Dex2jar :** Un Outil conçu pour effectuer le travail de conversion d'une application android au format dex à un fichier de format class Java.

— **Jd-gui :** Est un utilitaire graphique autonome qui affiche les codes source Java des fichiers «.class». (java).

— **JAD :** Décompilateur Java.

— **Dexdump :** Permet de décompiler les fichiers JAVA au format DEX [1].

— **Smali :** Assembleur / Désassembleur pour le format dex utilisé par dalvik [2].

Analyse dynamique :

La technique d'analyse dynamique comprend l'exécution de l'application sur une machine virtuelle ou un périphérique physique. Au milieu de l'examen, le comportement de l'application est surveillé et peut être disséqué. L'analyse dynamique donne une perspective d'application moins abstraite que l'analyse statique. Les chemins de code exécutés pendant l'exécution sont un sous-ensemble de chaque chemin accessible unique. Le principal objectif de l'analyse est de parvenir à une inclusion de code élevée, car chaque événement possible doit être activé pour surveiller tout comportement malveillant possible. Les principaux inconvénients de l'analyse dynamique sont les suivants : l'analyse dynamique nécessite des ressources considérables par rapport à l'analyse statique, ce qui l'empêche d'être distribuée sur des appareils mobiles à ressources limitées. De plus, l'analyse dynamique est responsable d'une couverture à faible code. Dernièrement, le malware a essayé de reconnaître l'émulateur et d'autres cadres d'analyse dynamique et de s'abstenir d'exposer leurs charges utiles[41].

1. Utilisé pour exécuter les applications développées pour l'OS Android
2. Est une machine virtuelle qui exécute les fichiers au format Dalvik Exécutable (.dex)

Outils d'analyse dynamique :

Il y a un certain nombre d'outils permettant de réaliser une analyse dynamique d'une application Android. On utilise ceux-ci pour tester des applications malveillantes dans un environnement protégé [18].

— **Droidbox :** Outil pour les applications Android de type Sandbox. Il permet l'analyse dynamique(moniteur d'API, analyse préliminaire statique, etc.).

— **Mobile Sandbox :** Outil pour les applications mobiles qui sont disponible en ligne.

Analyse hybride :

La technique d'analyse hybride comprend la consolidation des fonctionnalités statiques et dynamiques recueillies lors de l'examen de l'application et du dessin des données pendant que l'application est en cours d'exécution, séparément. Néanmoins, cela augmenterait la précision de l'identification. Le principal inconvénient de l'analyse hybride, il consomme les ressources du système Android et prend beaucoup de temps pour effectuer l'analyse [41].

2.4 Contre-mesures

La sécurité est primordiale et on doit mettre en place des systèmes permettant d'empêcher toutes les menaces extérieures et intérieures. Cette section Décrit un aperçu des contre-mesures qui peuvent être appliquées afin de réduire les risques d'attaques contre le système Android, parmi eux :

2.4.1 Antivirus

Les antivirus sont des logiciels de sécurité principalement utilisés sur les appareils mobiles. La popularité acquise sur les ordinateurs a contribué a accroître le niveau de confiance acquis par les utilisateurs mobiles. Avast, AVG et F-Secure sont des exemples d'antivirus renommés dans Android. Ils sont confrontés à de nouvelles contraintes apportées par l'évolution rapide des applications malveillantes. Comme les plates-formes de bureau, leur efficacité est étroitement liée à leurs méthodes de détection [48].

2.4.2 Pare-feu

L'utilisation d'un pare-feu sur les appareils mobiles Android n'est peut-être pas aussi critique qu'un PC, mais il peut être utile a gérer l'accès Internet pour une meilleure sécurité, une optimisation des données et des performances[7].

2.4.3 Les système de détection d'intrusions

Un système de détection d'intrusions (IDS) est un mécanisme de détection pour découvrir les tentatives de compromettre un système. Potentiellement, cela peut empêcher de telles tentatives. Dans ce cas, le système est appelé système de prévention des intrusions. Les mécanismes de détection des intrusions appliqués aux appareils mobiles Android sont basés sur les mêmes principes que les mécanismes utilisés dans d'autres systèmes (par exemple, les ordinateurs personnels et les réseaux informatiques). Même si les systèmes différents considérablement par leur type et leur architecture, les fondements de la protection contre les attaques restent les mêmes. Cela permet l'adoption des techniques et leur utilisation dans la zone de sécurité Android [20].

2.4.4 Analyse des logiciels malveillants

L'analyse des logiciels malveillants traite de l'étude du fonctionnement des logiciels malveillants et des résultats possibles de l'infection d'un logiciel malveillant spécifique donné. Il est important de savoir que les logiciels malveillants peuvent avoir différents fonctionnalités. Chaque fonctionnalité malveillante est conçu par les attaquants pour entrer dans le système à travers différentes sources pour infecter sans le consentement de l'utilisateur[6].

2.5 Conclusion

Nous avons présenté dans ce chapitre les limites de mécanismes de sécurité pour android. Ensuite nous avons défini les logiciels malveillants android, son cycle de vie, sa famille et ces techniques de détection. Enfin nous avons conclu avec les contre-mesures qui sont été appliquées afin de réduire les risques d'attaques contre ce système.

CHAPITRE 3

APPRENTISSAGE AUTOMATIQUE(ML) ET APPRENTISSAGE PROFOND(DL)

3.1 Introduction

L'Intelligence Artificielle est un domaine très vaste, où nous essayons d'imiter le comportement humain dans le but de rendre les machines si puissantes pour accomplir de nombreux types de tâches telles que la résolution de problèmes, la représentation des connaissances, la reconnaissance vocale, et d'autres. Grâce à ce domaine, et avec cette vague des techniques avancées d'apprentissage automatique et profond l'IA a fait un grand pas en avant.

Dans ce troisième chapitre nous définirons tout d'abord l'apprentissage automatique, ses différents types, ainsi que ses modèles les plus utilisé. Ensuite nous introduirons l'apprentissage profond en commençant par ses définitions, son histoire, ses différents modèles, ses métriques d'évaluation ainsi que ses fonctions d'activation. Enfin nous définirons les problèmes principaux de l'apprentissage profond et les différentes méthodes pour les éviter.

3.2 L'apprentissage automatique (ML)

3.2.1 Définitions

L'apprentissage automatique vu comme un domaine multidisciplinaire qui combine les statistiques et l'informatique. Plusieurs définitions sont données dans ce contexte :

L'apprentissage automatique a été inventé par Arthur Samuel,dés 1952 qui a créé le premier programme permettant de jouer et d'apprendre le jeu de dames. L'apprentissage automatique est la discipline qui donne à l'ordinateur la capacité d'apprendre sans qu'il soit explicitement programmé[27].

Nous disons qu'un programme apprend d'une expérience E par rapport à une classe de taches T et à une mesure de performance P, si sa performance aux taches en T mesuré par P, s'améliore avec l'expérience E [27]. L'apprentissage automatique est un domaine de recherche mature et reconnu, qui s'intéresse principalement a générer des modèles de prédiction qui sont utilisés pour découvrir des informations, des connaissances et d'autres régularités dans les données.

Le processus d'apprentissage est définit comme une combinaison de trois activités[24] :

— **La représentation :** Le modèle doit être représenté dans un langage formel que l'ordinateur peut le gérer.

— **L'évaluation :** Une fonction (fonction objective ou de notation) est nécessaire pour distinguer entre les bons modèles et les mauvais.

— **L'optimisation :** Une technique d'optimisation est fondamentale pour rechercher parmi les modèles le plus performant.

3.2.2 Les différents types d'apprentissage automatique

Apprentissage supervisé

L'apprentissage supervisé repose sur des informations utiles contenues dans des données étiquetées. La classification est la tâche la plus courante dans l'apprentissage supervisé. Cependant, l'étiquetage manuel des données est coûteux et prend du temps. Par conséquent, le manque de données étiquetées suffisantes constitue le principal goulot d'étranglement de l'apprentissage supervisé[22].

Apprentissage non supervisé

un apprentissage non supervisé extrait des informations précieuses sur les fonctionnalités des données non étiquetées, ce qui facilite considérablement l'obtention des données de formation. Cependant, les performances de détection des méthodes d'apprentissage non supervisé sont généralement inférieures à celles des méthodes d'apprentissage supervisé.[22]

Apprentissage semi-supervisé

Certains algorithmes peuvent traiter des données d'apprentissage partiellement étiquetées, généralement beaucoup de données non étiquetées et un peu de données étiquetées. C'est ce qu'on appelle l'apprentissage semi-supervisé [26].

L'apprentissage semi-supervisé (SSL) est un type de technique d'apprentissage automatique (ML). Il est à mi-chemin entre l'apprentissage supervisé et non supervisé. L'objectif principal de SSL est de surmonter les inconvénients de l'apprentissage supervisé et non supervisé. L'apprentissage supervisé nécessite une énorme quantité de données de formation pour classer les données de test, ce qui est un processus rentable et long. D'un autre côté, l'apprentissage non supervisé ne nécessite aucune donnée étiquetée, qui regroupe les données en fonction de la similitude des

points de données en utilisant une approche de clustering ou de probabilité maximale. Principal inconvénient de cette approche, elle ne peut pas regrouper avec précision des données inconnues. Pour surmonter ces problèmes, SSL a été proposé par la communauté des chercheurs, qui peut apprendre avec une petite quantité de données de formation peut étiquetée. SSL crée un modèle avec quelques modèles étiquetés comme données d'apprentissage et traite le reste des modèles comme des données de test [21].

La plupart des algorithmes d'apprentissage semi-supervisés sont des combinaisons d'algorithmes non supervisés et supervisés. Par exemple, les réseaux de croyances profondes (DBN) sont basés sur des composants non supervisés appelés machines Boltzmann restreintes (RBM) empilées les unes sur les autres. Les RBM sont formés séquentiellement de manière non supervisée, puis l'ensemble du système est affiné à l'aide de techniques d'apprentissage supervisé [26].

Apprentissage par renforcement

L'apprentissage par renforcement est une technique très différente. Le système d'apprentissage, appelé agent dans ce contexte, peut observer l'environnement, sélectionner et exécuter des actions et obtenir des récompenses en retour. Il doit ensuite apprendre par ces propres erreurs quelle est la meilleure stratégie, appelée politique, pour obtenir le plus de récompenses au fil du temps. Une politique définit l'action que l'agent doit choisir lorsqu'il se trouve dans une situation donnée [26].

3.2.3 Les modèles d'apprentissage automatique

Nous présentons dans ce qui suit un résumé des modèles de classification les plus utilisés dans la détection des malwares Android, ainsi que les avantages et les inconvénients des modèles d'apprentissage supervisé et non supervisé(Tableau 3.1).

Réseau de neurones artificiels (ANN) :

L'idée de conception d'un ANN est d'imiter le fonctionnement du cerveau humain. Un ANN contient une couche d'entrée, plusieurs couches cachées et une couche de sortie. Un ANN contient un grand nombre d'unités et peut théoriquement approximer des fonctions arbitraires. Par conséquent, il a une forte capacité d'adaptation, en particulier pour les fonctions non linéaires. La formation des ANN prend du temps à cause de la structure complexe du modèle. Il est a noter que les modèles ANN sont formés par l'algorithme de rétropropagation qui ne peut pas être utilisé pour former des réseaux profonds. Ainsi, un ANN appartient à des modèles peu profonds et diffère des modèles d'apprentissage profond[22].

Machine à vecteurs de support (SVM) :

Il s'agit d'un algorithme puissant utilisé pour les problèmes de classification et de régression. Il est basé sur la représentation de chaque exemple dans un espace à n dimensions, où un hyperplan

est créé en poursuivant la meilleure séparation entre les classes. Pour construire cet hyperplan, une partie des données d'apprentissage appelées vecteurs de support est utilisée. Ces modèles ont été largement utilisés pour créer des outils de détection de logiciels malveillants[38].

Les k plus proches voisins(KNN) :

Le K-plus proche voisin est un classifieur non paramétrique, ce qui signifie que ce modèle croît parallèlement à la taille des données d'apprentissage. Fondamentalement, il utilise une métrique de distance pour fournir une prédiction basée sur la classe majoritaire parmi les K points les plus proches de l'exemple. Bien que ces modèles se soient révélés puissants dans des problèmes variés, ils ne sont pas indiqués pour les espaces de grande dimension[38].

Le classifieur naïf de Bayes :

Le classificateur Naïve Bayes est un modèle probabiliste qui tombe sous le coup de l'apprentissage bayésien. Il est basé sur le théorème de Bayes et cherche a calculer l'hypothèse avec une probabilité plus élevée d'un espace défini par des données d'entraînement. Ces modèles se caractérisent par leur simplicité mais aussi par leur capacité avérée a traiter des problèmes variés.[38]

Arbre de décision :

L'algorithme d'arbre de décision classe les données à l'aide d'une série de règles. Le modèle ressemble à un arbre, ce qui le rend interprétable. L'algorithme d'arbre de décision peut automatiquement exclure les fonctionnalités non pertinentes et redondantes. Le processus d'apprentissage comprend la sélection des fonctionnalités, la génération d'arbres et l'élagage des arbres. Lors de la formation d'un modèle d'arbre de décision, l'algorithme sélectionne individuellement les fonctionnalités les plus appropriées et génère des nœuds enfants à partir du nœud racine[38].

Clustering :

Le clusering est basé sur la théorie de la similitude, c'est-à-dire le regroupement de données hautement similaires dans les mêmes clusters et le regroupement de données moins similaires dans différents clusters. Différent de la classification, le clustering est un type d'apprentissage non supervisé, Par conséquent, les exigences relatives aux ensembles de données sont relativement faibles. Cependant, lors de l'utilisation d'algorithmes de clustering pour détecter des attaques, il est nécessaire de référencer des informations externes[22].

K-means :

K-means est un algorithme de clustering typique, où K est le nombre de clusters et la moyenne est la moyenne des attributs. L'algorithme K-means utilise la distance comme critère de mesure de similarité. Plus la distance entre deux objets de données est courte, plus ils sont susceptibles

d'être placés dans le même cluster. L'algorithme K-means s'adapte bien aux données linéaires, mais ses résultats sur les données non convexes ne sont pas idéaux. De plus, l'algorithme K-means est sensible à la condition d'initialisation et au paramètre K. Par conséquent, de nombreuses expériences répétées doivent être exécutées pour définir la valeur du paramètre appropriée[22].

Comparaison de différents modèles d'apprentissage automatique :

Algorithmes	Avantages	Inconvénient
ANN	Capable de traiter des données non linéaires ; Forte capacité d'adaptation ;	Convient au sur-apprentissage ; A tendance a rester coincé dans un optimum local ; La formation des modèles prend du temps ;
KNN	Appliquer à des données massives ; Convient aux données non linéaires ; Entraînement rapide ; Robuste au bruit ;	Faible précision sur la classe minoritaire ; Temps de test lent ; Sensible au paramètre K
SVM	Apprenez des informations utiles à partir d'un petit train ; Forte capacité de génération ;	Ne fonctionne pas bien sur les méga données ou les tâches de classification multiples ; Sensible aux paramètres des fonctions du noyau ;
Naïve Bayes	Robuste au bruit ; Capable d'apprendre progressivement ;	Ne donne pas de bons résultats sur les données liées aux attributs ;
Arbre de décision	Sélectionnez automatiquement les fonctionnalités ; Interprétation forte ;	Tendances des résultats de la classification à la classe majoritaire ; Ignorer la corrélation des données ;
K-means	Simple, peut être formé rapidement ; Grande évolutivité ; Peut s'adapter aux méga données ;	Ne fonctionne pas bien sur des données non convexes ; Sensible à l'initialisation ; Sensible au paramètre K ;

TABLE 3.1 – Les avantages et les inconvénients des algorithmes d'apprentissage automatique[38].

Les modèles traditionnels d'apprentissage automatique (modèles peu profonds) pour la détection de logiciels malveillants comprennent principalement le réseau de neurones artificiels (ANN), la machine à vecteur de support (SVM), le voisin K le plus proche (KNN), les naïfs Bayes, la régression logistique (LR), l'arbre de décision, le regroupement et méthodes combinées et hybrides.

Certaines de ces méthodes sont étudiées depuis plusieurs décennies et leur méthodologie est mature. Ils se concentrent non seulement sur l'effet de détection mais également sur des problèmes pratiques, par exemple l'efficacité de la détection et la gestion des données [38].

3.3 L'apprentissage profond (DL)

3.3.1 Définitions

L'apprentissage profond vu comme l'une des principales technologies d'apprentissage automatique et d'intelligence artificielle.la figure 3.1 démontre la relation entre ces derniers. Plusieurs définitions sont données pour l'apprentissage profond :

Définition 01 : L'apprentissage profond est un ensemble d'algorithmes d'apprentissage automatique qui utilise plusieurs couches qui correspondent à différents niveaux d'abstraction à chaque niveau. Il se compose d'une couche d'entrée, d'une couche de sortie et de plusieurs couches cachées. Il est utilisé pour la synthèse vocale, le traitement d'image, la reconnaissance de l'écriture manuscrite, la détection d'objets, l'analyse de prédiction et la prise de décision [23].

Définition 02 : L'apprentissage profond permet aux modèles de calcul composés de plusieurs couches de traitement d'apprendre des représentations de données avec plusieurs niveaux d'abstraction. Ces méthodes ont considérablement amélioré l'état de l'art en matière de reconnaissance vocale, de reconnaissance visuelle d'objets, de détection d'objets et de nombreux autres domaines tels que la découverte de médicaments et les logiciels malveillants. [36].

Définition 03 : L'apprentissage profond est une branche d'apprentissage automatique qui dépend de l'étude de différents niveaux de représentations, analogues a un classement de caractéristiques ou de notions, où les notions de haut niveau sont déterminées à partir de celles de niveau inférieur, et des notions similaires de niveau inférieur pourraient aider a déterminer de nombreuses notions de haut niveau [42].

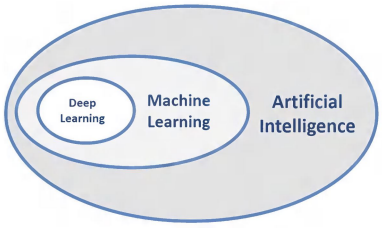


FIGURE 3.1 – La relation entre l'intelligence artificielle, l'apprentissage automatique et l'apprentissage profond [1].

3.3.2 Histoire de l'apprentissage profond

Année	Contributeur	Contribution
1943	McCulloch and Pitts	introduction du McCulloch–Pitts (MCP) modèle considérer comme L'ancêtre des réseaux de neurones artificielles
1949	Donald Hebb	Il a introduit l'apprentissage de Hebb qui servira de fondation pour les réseaux de neurones modernes
1958	Frank Rosenblatt	introduction du premier perceptron
1974	Paul Werbos	Il a introduit la rétro propagation
1980	Teuvo Kohonen	Il a introduit les cartes auto organisatrices
1980	Kunihiko Fukushima	introduction du Neocognitron, qui a inspiré les réseaux de neurones convolutif
1982	John Hopfield	Il a introduit des réseaux de Hopfield
1985	Hilton and Sejnowski	introduction des machines de Boltzmann
1986	Paul Smolensky	Il a introduit la règle de Harmonium, qui sera connu plus tard comme machines de Boltzmann restreintes
1986	Michael I. Jordan	présentation des réseaux de neurones récurrent
1990	Yann LeCun	introduction de LeNet et montra la capacités des réseaux de neurones profond
2006	Geoffrey Hinton	Il a introduit le Deep belief Network
2009	Salakhutdinov and Hinton	introduction des Deep Boltzmann Machines
2012	Alex Krizhevsky	introduction de AlexNet qui remporta le challenge ImageNet
2015	Google DeepMind	le programme AlphaGo, capable de jouer au jeu de go grâce à la méthode de l'apprentissage profond.
2016	Google DeepMind	le programme AlphaGo bat le champion du monde.

Table 3.2 – Les étapes majeurs d'apprentissage profond [50].

3.3.3 Les différents modèles d'apprentissage profond

Cette section démontre les différents types d'apprentissage profond et résume les algorithmes de classification de ces types les plus utilisés dans la détection des malwares Android, ainsi nous présentons les avantages et les inconvénients des algorithmes d'apprentissage profond.

Machine Boltzmann restreinte (RBM) :

Un RBM est un réseau neuronal randomisé dans lequel les unités obéissent à la distribution de Boltzmann. Un RBM est composé d'une couche visible et d'une couche cachée. Les unités d'une même couche ne sont pas connectées. Cependant, les unités dans les différentes couches sont entièrement connectées, comme le montre la figure3.2. où $\mathbf{v_i}$ est une couche visible et $\mathbf{h_i}$ est une couche cachée. Les RBM ne font pas de distinction entre les directions avant et arrière. Ainsi, les poids dans les deux directions sont les mêmes. Les RBM sont des modèles d'apprentissage non supervisés formés par l'algorithme de divergence contrastive, et ils sont généralement appliqués pour l'extraction de caractéristiques [38].

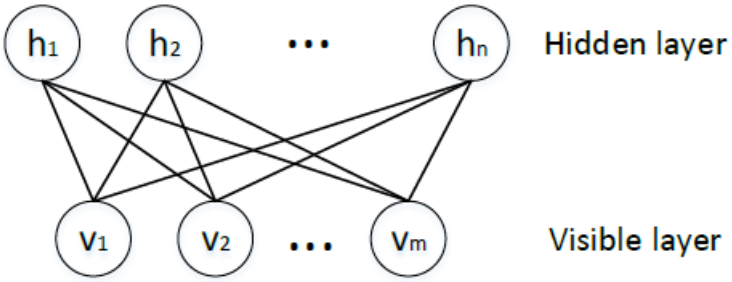


FIGURE 3.2 – La structure du RBM [38].

Réseau de croyances profondes (DBN) :

Un DBN se compose de plusieurs couches RBM et d'une couche de classification softmax, comme le montre la figure 3.3. La formation d'un DBN comprend deux étapes : une pré-formation non supervisée et un réglage fin supervisé. Tout d'abord, chaque RBM est formé à l'aide d'un prétraitement gourmand en couches. Ensuite, le poids de la couche softmax est appris par des données étiquetées. Dans la détection d'attaque, les DBN sont utilisés pour l'extraction et la classification des fonctionnalités [38].

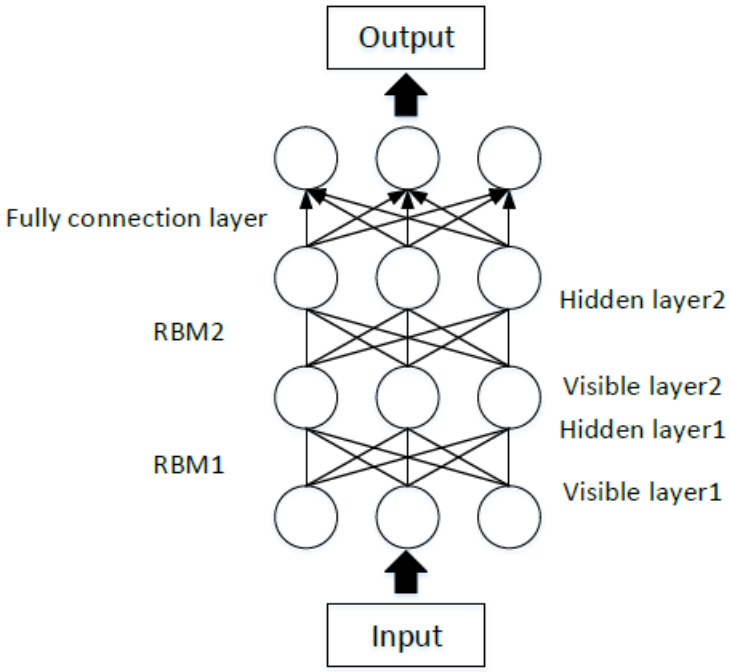


FIGURE 3.3 – La structure du DBN [38].

Réseau de neurones profonds (DNN) :

Une stratégie de pré-formation et de réglage fin au niveau des couches permet de construire des DNN avec plusieurs couches, comme le montre la figure 3.4. Lors de la formation d'un DNN, les paramètres sont d'abord appris à l'aide de données non étiquetées, qui est une étape d'apprentissage des fonctionnalités non supervisées. Ensuite, le réseau est réglé via les données étiquetées, qui est une étape d'apprentissage supervisé. Les réalisations étonnantes des DNN sont principalement dues à l'étape d'apprentissage des fonctionnalités non supervisées [38].

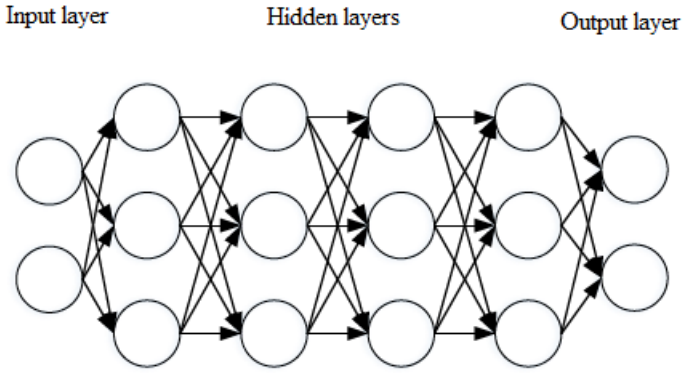


FIGURE 3.4 – La structure du DNN [38].

Réseau neuronal convolutif (CNN) :

Les CNN sont conçus pour imiter le système visuel humain (HVS). Par conséquent, les CNN ont atteint de grandes réalisations dans le domaine de la vision par ordinateur. Un CNN est empilé avec des couches de convolution d'une dimension, deux dimensions et trois dimensions (Conv1D,Conv2D et Conv3D) et de mise en commun alternatives (Maxpooling1D,Maxpooling2D et Maxpooling3D), comme le montre la figure 3.5. Les couches convolutives sont utilisées pour extraire des entités et les couches de mise en commun sont utilisées pour améliorer la généralisabilité des entités [38]. Lorsque ces couches sont empilées, une architecture CNN a été formée.

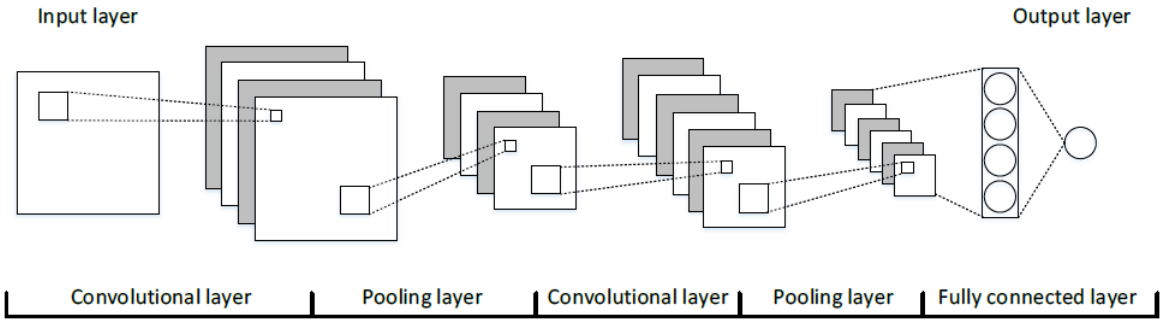


FIGURE 3.5 – La structure du CNN [38].

Auto-encodeur (DAE) :

Un auto-encodeur est un réseau de neurones artificiels qui possède deux composants symétriques(encodeur et décodeur), comme illustré dans la figure 3.6. L'encodeur extrait des fonctionnalités à partir de données brutes et le décodeur reconstruit les données à partir des fonctionnalités extraites. Pendant l'entraînement, la divergence entre l'entrée du codeur et la sortie du décodeur est progressivement réduite. Lorsque le décodeur réussit à reconstruire les données via les fonctionnalités extraites, cela signifie que les fonctionnalités extraites par le codeur représentent l'ensemble de données. Il est important de noter que l'ensemble de ce processus ne nécessite aucune information supervisée [38].

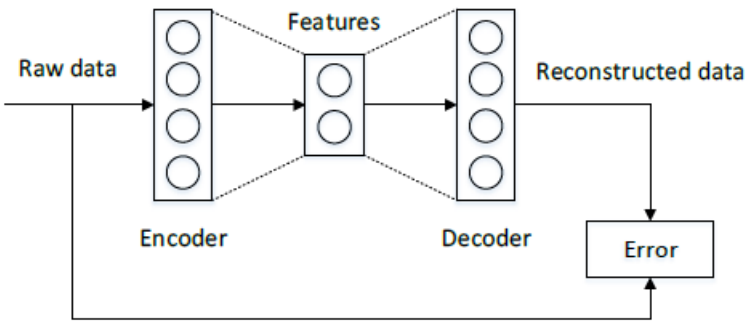


FIGURE 3.6 – La structure d'un auto-encodeur [38].

Réseau de neurones récurrents (RNN) :

Les RNN sont des réseaux conçus pour des données séquentielles et sont largement utilisés dans le traitement du langage naturel (PNL). Les caractéristiques des données séquentielles sont contextuelles. L'analyse de données isolées de la séquence n'a aucun sens. Pour obtenir des informations contextuelles, chaque unité d'un RNN reçoit non seulement l'état actuel mais également les états précédents. La structure d'un RNN est illustrée à la figure 3.7. Lorsque tous les éléments **W** de la figure 3.7 sont identiques. Cette caractéristique fait que les RNN souffrent souvent de gradients qui disparaissent ou explosent. En réalité, les RNN standard ne traitent que des séquences de longueur limitée. Pour résoudre le problème de dépendance à long terme, de nombreuses variantes de RNN ont été proposées, telles que la mémoire à court terme à long terme (LSTM), l'unité récurrente fermée (GRU) [38].

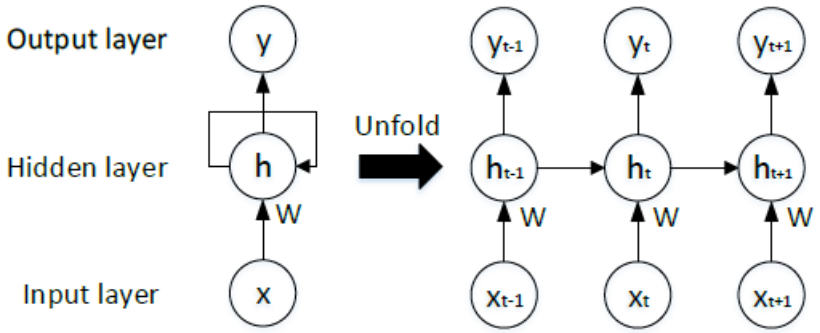


FIGURE 3.7 – La structure d'un RNN [38].

— **LSTM(Long Short Term Memory) :** Les LSTM sont un type particulier de RNN, capable d'apprendre des dépendances à long terme. Les LSTM sont explicitement conçus pour éviter le problème de dépendance à long terme. Se souvenir des informations pendant de longues périodes. Ils fonctionnent extrêmement bien sur une grande variété de problèmes et sont maintenant largement utilisés [4].

— **GRU(unité récurrente fermée) :** L'unité récurrente fermée est un autre type de cellule RNN, vise a résoudre le problème du gradient de fuite qui vient avec un réseau de neurones récurrent standard. GRU est une modification de la couche cachée RNN qui permet de mieux capturer les connexions à longue portée. GRU est similaire au LSTM mais sa structure est simplifiée. Comme LSTM, pour résoudre le problème de gradient de fuite d'un RNN standard, GRU utilise, soi-disant, la porte de mise à jour et la porte de réinitialisation. Fondamentalement, ce sont deux vecteurs qui décident quelles informations doivent être transmises à la sortie[4].

Comparaison des différents modèles d'apprentissage profond

Algorithmes	Aventages	Inconvénients
RBM	- Permet de produire des échantillons comme s'ils provenaient de la distribution des données - Il peut être utilisé comme extracteur de fonctionnalités pour entraîner d'autres modèles par-dessus	- Difficile de bien s'entraîner - Le calcul de la probabilité prend du temps
DBN	Offrir une approche d'apprentissage couche par couche pour initialiser le réseau	La phase de formation consomme des ressources système en raison du processus d'initialisation et d'échantillonnage.
CNN	- Moins de connexions neuronales nécessaires par rapport à un NN standard. - De nombreuses variantes de CNN ont été développées	- Habituellement, il a besoin de plusieurs couches pour découvrir une hiérarchie complète des fonctionnalités visuelles - Il a généralement besoin d'un grand ensemble de données d'images balisées
RNN	- Modélisation des dépendances temporelles - Capable de se souvenir des événements en série	Le processus d'apprentissage souffre d'un problème de gradient qui disparaît (un grand changement dans la valeur des paramètres pour les premières couches n'a pas un grand effet sur la sortie)
DAE	- Appliqué à la fonction d'extraction / réduction de dimensionnalité. - De nombreuses variantes de DAE ont été proposées	- Il nécessite une étape de pré-formation - Il n'a pas la capacité de déterminer quelles données sont pertinentes
DNN	Succès accompli dans différentes applications	Le processus d'apprentissage pourrait prendre du temps

TABLE 3.3 – Les avantages et les inconvénients des algorithmes d'apprentissage profond [41].

Les modèles d'apprentissage profond se composent de divers réseaux profonds. Parmi eux, les réseaux profonds et brefs (DBN), les réseaux de neurones profonds (DNN), les réseaux de neurones convolutifs (CNN) et les réseaux de neurones récurrents (RNN) sont des modèles d'apprentissage supervisé, tandis que les auto-encodeurs(DAE), les machines Boltzmann restreintes (RBM) sont des modèles d'apprentissage non supervisés. Le nombre d'études sur les système de détection des malwares android basés sur l'apprentissage profond a augmenté rapidement de 2015 à nos jours. Les modèles d'apprentissage profond apprennent directement les représentations d'entités à partir des données originales, telles que des images et des textes, sans nécessiter une ingénierie manuelle des entités. Ainsi, les méthodes d'apprentissage profond peuvent s'exécuter de bout en bout. Pour les grands ensembles de données, les méthodes d'apprentissage profond ont un avantage significatif sur les modèles peu profonds. Dans l'étude de l'apprentissage profond, les principaux axes sont l'architecture de réseau, la sélection des hyperparamètres et la stratégie d'optimisation [38].

3.3.4 Métriques d'évaluation

De nombreuses métriques sont utilisées pour évaluer les méthodes d'apprentissage automatique et d'apprentissage profond. Les modèles optimaux sont sélectionnés à l'aide de ces métriques [38] :

Accuracy :

Est une mesure appropriée lorsque l'ensemble de données est équilibré. Dans des environnements réseau réels ; cependant, les échantillons normaux sont beaucoup plus abondants que les échantillons anormaux ; par conséquent, Accuracy peut ne pas être une mesure appropriée.

$$Accuracy = \frac{TP + TN}{TP + FP + FN + TN} \tag{3.1}$$

Précision (P) :

Est défini comme le rapport des échantillons positifs réels aux échantillons positifs prédits, il représente la confiance de la détection d'attaque.

$$P = \frac{TP}{TP + FP} \tag{3.2}$$

Rappel (R) :

Est défini comme le rapport des vrais échantillons positifs au total des échantillons positifs et est également appelé le taux de détection. Le taux de détection reflète la capacité du modèle à reconnaître les attaques, qui est une mesure importante dans les systèmes de détection de malwares.

$$R = \frac{TP}{TP + FN} \tag{3.3}$$

Mesure-F (F) :

Est définie comme la moyenne pondérée de la précision et du rappel.

$$F = \frac{2 * P * R}{P + R} \tag{3.4}$$

Le taux de faux négatifs (FNR) :

est défini comme le rapport des échantillons faussement négatifs au total des échantillons positifs. Dans la détection d'attaque, le FNR est également appelé le taux d'alarme manquée.

$$FNR = \frac{FN}{TP + FN} \tag{3.5}$$

Le taux de faux positifs (FPR) :

est défini comme le rapport des échantillons faux positifs aux échantillons positifs prédits. Dans la détection d'attaque, le FPR est également appelé le taux de fausse alarme, et il est calculé comme suit :

$$FPR = \frac{FP}{TN + FP} \tag{3.6}$$

Où TP est le vrai positif(Nombre d'échantillons correctement distingué comme malveillant), FP est le faux positif(Nombre d'échantillons identifiés à tort comme malveillants), TN est le vrai négatif-(Nombre d'échantillons correctement distingués comme bénins), FN est le faux négatif(Nombre d'échantillons identifiés à tort comme bénins).

3.3.5 Les fonctions d'activation

Les classificateurs de l'apprentissage profond sont constitué de plusieurs couches, chaque couche a plusieurs neurones, la fonction utilisée dans un neurone est généralement appelée fonction d'activation.

Les fonctions d'activation sont des fonctions utilisées dans les réseaux de neurones pour calculer la somme pondérée des entrées et des biais, qui sert à décider si un neurone peut être déclenché ou non. Il manipule les données présentées à travers un traitement de gradient généralement descente de gradient et produit ensuite une sortie pour le réseau neuronal, qui contient les paramètres dans les données [44]. À ce jour, 7 fonctions d'activation majeures ont été essayées, parmi eux les 4 fonctions d'activation les plus utilisé : sigmoïde, tanh, ReLU et softmax. Chacun d'eux est décrit en détail ci-dessous. la figure suivante représente une fonction d'activation dans un neurone :

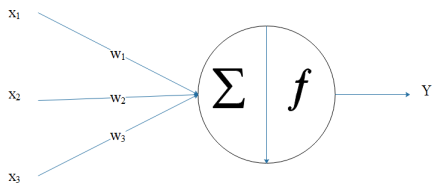


FIGURE 3.8 – fonction d'activation dans un neurone.

Fonction sigmoïde

La fonction sigmoïde est une fonction d'activation non linéaire utilisé principalement dans les réseaux de neurones à action directe. Il s'agit d'une fonction réelle différentiable bornée, définie pour des valeurs d'entrée réelles, avec des dérivées positives partout et un certain degré de régularité. La fonction sigmoïde est donnée par :

$$f(x) = (\frac{1}{(1 + e^{-x})}) \tag{3.7}$$

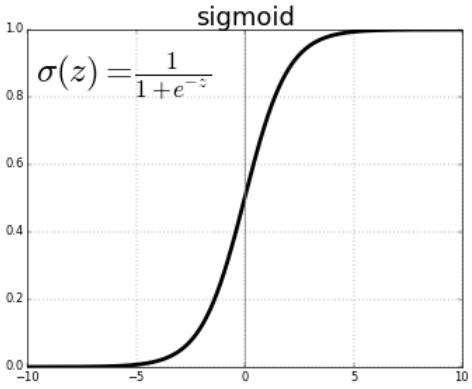


FIGURE 3.9 – Fonction Sigmoïde.

Fonction tangente hyperbolique (Tanh)

La fonction tanh est un autre type d'AF utilisé en DL et il a quelques variantes utilisées dans les applications DL. La fonction tangente hyperbolique connue sous le nom de fonction tanh, est une fonction centrée sur zéro plus douce dont la plage se situe entre -1 et 1, ainsi la sortie de la fonction tanh est donnée par :

$$f(x) = (\frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}) \tag{3.8}$$

La fonction tanh est devenue la fonction préférée par rapport à la fonction sigmoïde en ce qu'elle donne de meilleures performances d'entraînement pour les réseaux de neurones multicouches.

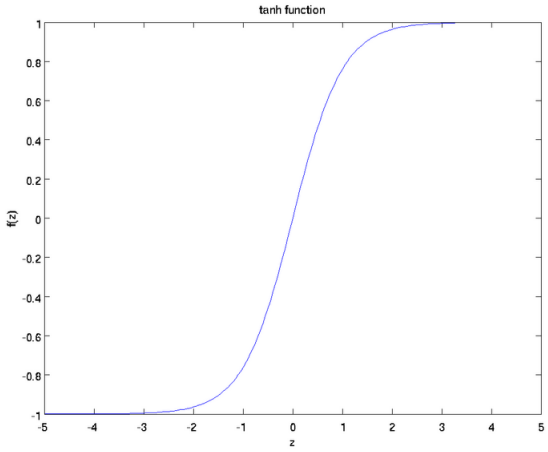


FIGURE 3.10 – Fonction Tanh.

Fonction Softmax

La fonction Softmax est un autre type de AF utilisé dans le calcul neuronal. Il est utilisé pour calculer la distribution de probabilité à partir d'un vecteur de nombres réels. La fonction Softmax produit une sortie qui est une plage de valeurs entre 0 et 1, la somme des probabilités étant égale à 1. La fonction Softmax est calculée en utilisant la relation :

$$f(x_i) = (\frac{\exp(c_i)}{\sum_j \exp(x_j)}) \tag{3.9}$$

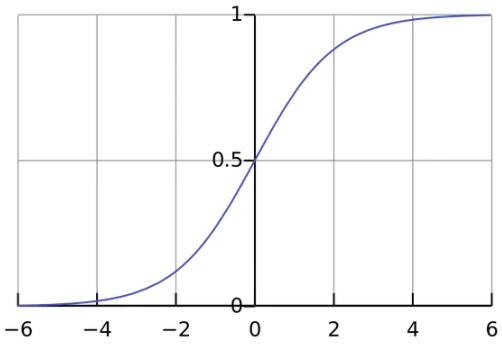


FIGURE 3.11 – Fonction Softmax.

Fonction d'unité linéaire rectifiée (ReLU)

La fonction Relu est un AF à apprentissage plus rapide, qui s'est avéré d'être la fonction la plus efficace et la plus utilisée. Il offre les meilleures performances et généralisation en apprentissage profond par rapport aux fonctions d'activation Sigmoid et tanh. Le ReLU représente une fonction presque linéaire et préserve donc les propriétés des modèles linéaires qui les ont rendus faciles

à optimiser, avec la méthode de gradient-descente. La fonction d'activation ReLU effectue une opération de seuil pour chaque élément d'entrée où les valeurs inférieures à zéro sont mises à zéro, ainsi ReLU est donné par :

$$f(x) = \max(0, x) \tag{3.10}$$

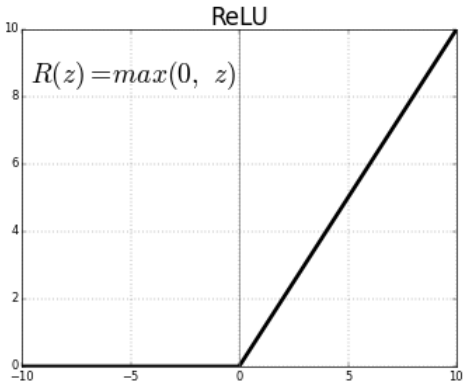


FIGURE 3.12 – Fonction ReLU.

3.3.6 Sur-apprentissage et Sous-aapprentissage(Overfitting and Underfitting)

Les problèmes principaux de l'apprentissage profond est de savoir comment créer un modèle qui fonctionne bien sur l'ensemble d'apprentissage et l'ensemble de test. De nombreuses stratégies utilisées dans l'apprentissage profond sont explicitement conçues pour réduire l'erreur de test, éventuellement au détriment d'une augmentation de l'erreur d'apprentissage. Ces stratégies sont connues sous le nom de sur-apprentissage et sous-apprentissage.

Sur-apprentissage

L'un des plus gros problème dans la formation des réseaux de neurones est le sur-apprentissage des données de formation. Cela signifie que le réseau de neurones à un certain moment pendant la période d'entraînement n'améliore plus sa capacité à résoudre le problème. Mais commence tout juste à apprendre une certaine régularité aléatoire contenue dans l'ensemble des modèles de formation. Cela équivaut à l'observation empirique que l'erreur sur l'ensemble de test a un minimum où la capacité de généralisation du réseau est la meilleure avant que cette erreur ne recommence à augmenter [31]. La figure 3.13 démontre un example de sur-apprentissage.

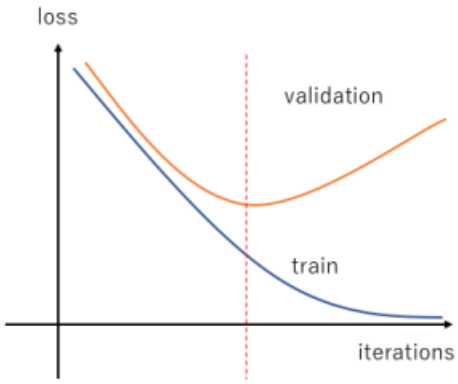


FIGURE 3.13 – Example de sur-apprentissage de perte de validation et perte de formation.

Sous-apprentissage

Est l'opposite du sur-apprentissage. Cela se produit lorsque le modèle est incapable de saisir la variabilité des données. Le classificateur résultant n'aura aucun pouvoir prédictif et ne sera pas en mesure de cartographier correctement les données de formation.Ceci est le résultat de la compréhension ou de la tentative d'utiliser un modèle trop simple pour décrire un ensemble de données donné [31]. La figure 3.14 démontre un example de sous-apprentissage.

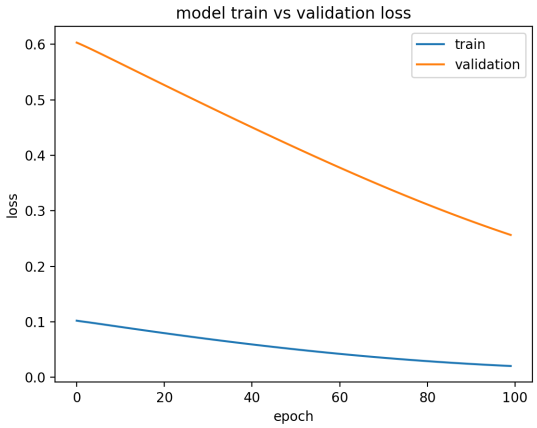


FIGURE 3.14 – Example de sous-apprentissage de perte de validation et perte de formation.

Quelques méthodes pour éviter le problème du sur-apprentissage et sous-apprentissage

Il existe de nombreuses méthodes pour éviter le problème du sur-apprentissage et sous-apprentissage parmi eux :

— **Arrêt anticipé(Early-stopping) :** Cette stratégie permet d'éviter le phénomène de «ralentissement de la vitesse d'apprentissage». Ce problème signifie que la précision des algorithmes cesse de s'améliorer après un certain point, Comme le montre la figure 3.15, où l'axe

horizontal est l'époque et l'axe vertical est l'erreur, la ligne bleue montre l'erreur d'apprentissage et la ligne rouge montre l'erreur de validation. Si le modèle continue à apprendre après le point, l'erreur de validation augmentera tandis que l'erreur d'apprentissage continuera à diminuer. Si nous arrêtons d'apprendre avant le point, c'est insuffisant. Si nous nous arrêtons après le point, nous obtenons un sur-apprentissage. L'objectif est donc de trouver le point exact pour arrêter la formation [52].

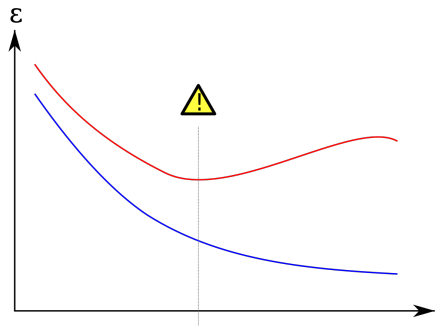


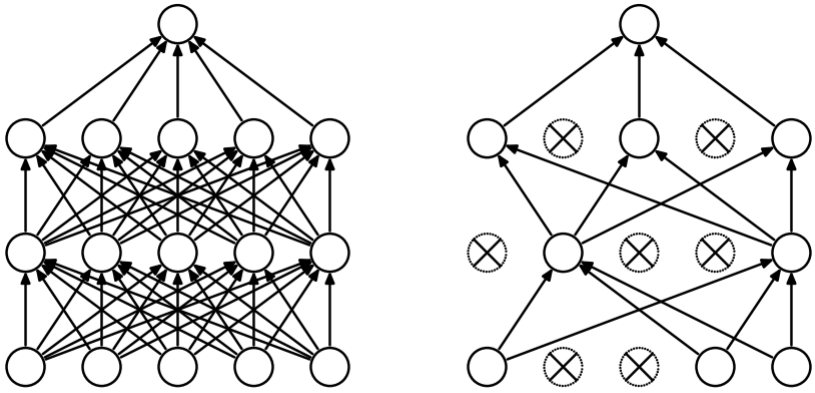
FIGURE 3.15 – Erreur de validation vs erreur de test [52].

— **Réduction du réseau :** L'apprentissage du bruit est une cause importante de sur-apprentissage. Donc, logiquement, la réduction du bruit devient une voie de recherche pour l'inhibition du sur-ajustement. Sur la base de cette réflexion, l'élagage est proposé pour réduire la taille des classificateurs finaux dans l'apprentissage relationnel, en particulier dans l'apprentissage par arbre de décision. L'élagage est une théorie importante utilisée pour réduire la complexité de la classification en éliminant les données moins significatives ou non pertinentes, et enfin pour empêcher le sur-apprentissage et améliorer la précision de la classification [52].

— **Régularisation :** Généralement, la sortie d'un modèle peut être affectée par plusieurs fonctionnalités. Lorsque le nombre de fonctionnalités augmente, le modèle devient compliqué. Un modèle surajusté a tendance à prendre en considération toutes les caractéristiques, même si certaines d'entre elles ont un effet très limité [52].

Il existe plusieurs méthodes générales de régularisation. Nous allons présenté l'une de ces méthodes de régularisation que nous allons utilisé a la suite qui est une technique très puissante.

- **Dropout :** Le Dropoute est une technique populaire et efficace contre le sur-apprentissage dans les réseaux de neurones. L'idée initiale du Dropout est de supprimer au hasard des unités et des connexions pertinentes des réseaux de neurones pendant l'entraînement. Cela empêche les unités de trop s'adapter [52], Comme le montre la figure 3.16.



(a) Standard Neural Net (b) After applying dropout.

FIGURE 3.16 – Illustration du dropout lors de l'apprentissage (à droite) et lors du test (à gauche)[32].

3.4 Conclusion

Nous avons présenté en détails les différentes techniques d'apprentissage automatique et d'apprentissage profond ainsi que ces métriques d'évaluation et ces fonctions d'activations. Nous avons conclu avec ces différents problèmes pour les éviter. Le chapitre suivant sera consacrer sur les publications et les travaux utilisés pour la détection des malwares android.

CHAPITRE 4

APRENTISSAGE AUTOMATIQUE ET PROFOND POUR LA DÉTECTION DES MALWARES ANDROID

4.1 Introduction

Diverses publications se sont concentrées sur l'utilisation de techniques d'apprentissage automatique et d'apprentissage profond pour mettre en œuvre des mécanismes de détection et de classification des logiciels malveillants. Actuellement, il existe plusieurs approches pour les applications Android.

Nous présentons dans ce chapitre quelques approches crées pour détecter les malwares Android en utilisant les techniques d'apprentissage automatique et profond.

4.2 L'apprentissage automatique pour la détection des malwares Android

4.2.1 DREBIN

Arp, Daniel Spreitzenbarth, Michael Hübner, Malte Gascon ,HugoRieck et Konrad [15], ont proposé DREBIN : Détection efficace et explicite des logiciels malveillants Android dans votre poche, une méthode légère de détection des malwares Android qui déduit automatiquement les schémas de détection et permet d'identifier les malwares directement sur le smartphone. DREBIN effectue une large analyse statique, rassemblant autant de fonctionnalités du code et du manifeste d'une application que possible. Ces fonctionnalités sont organisées en ensembles de chaînes

(telles que les autorisations, les appels d'API et les adresses réseau) et intégrées dans un espace vectoriel commun. Par exemple, une application qui envoie des SMS premium est castée dans une région spécifique de l'espace vectoriel associée aux autorisations, intentions et appels API correspondants. Cette représentation géométrique permet à DREBIN d'identifier automatiquement les combinaisons et les schémas de caractéristiques indicatives pour les logiciels malveillants à l'aide de techniques d'apprentissage automatique. Pour chaque application détectée, les modèles respectifs peuvent être extraits, mis en correspondance avec des descriptions significatives, puis fournis à l'utilisateur comme explication de la détection. Outre la détection, DREBIN peut également fournir des informations sur les échantillons de logiciels malveillants identifiés [15].

Fonctionnement de l'outil

Pour détecter les logiciels malveillants sur un smartphone, DREBIN nécessite une représentation complète mais légère des applications qui permet de déterminer les indications typiques d'une activité malveillante. Cette méthode utilise une large analyse statique qui extrait des ensembles de fonctionnalités de différentes sources et les analyse dans un espace vectoriel expressif. Ce processus est illustré dans la figure 4.1 et décrit dans ce qui suit [15] :

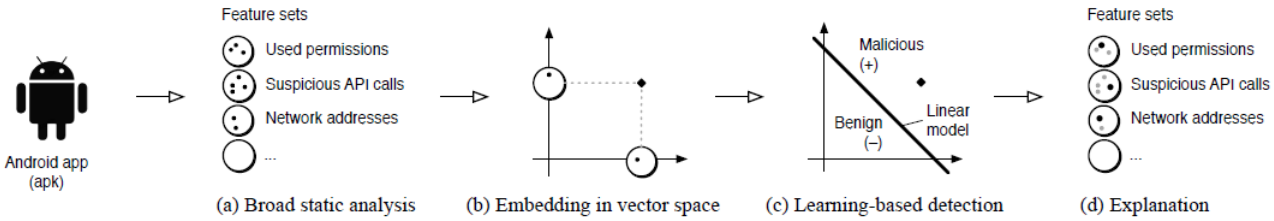


FIGURE 4.1 – Représentation schématique des étapes d'analyse effectuées par Drebin [15].

1. **Large analyse statique :** Dans la première étape, DREBIN inspecte statiquement une application Android donnée et extrait différents ensembles de fonctionnalités du code manifeste et dex de l'application.
2. **Incorporation dans l'espace vectoriel :** Les ensembles d'entités extraits sont ensuite mappés à un espace vectoriel commun, où les motifs et les combinaisons des entités peuvent être analysés géométriquement.
3. **Détection basée sur l'apprentissage :**L'intégration des ensembles de fonctionnalités nous permet d'identifier les logiciels malveillants à l'aide de techniques efficaces d'apprentissage automatique, telles que les machines à vecteurs de support linéaires.
4. **Explication :** Dans la dernière étape, les fonctionnalités contribuant à la détection d'une application malveillante sont identifiées et présentées à l'utilisateur pour expliquer le processus de détection.

Les limitations

— Il ne peut généralement pas interdire les infections par des applications malveillantes.

— Manque d'analyse dynamique.

— les attaques de transformation sont non détectables par analyse statique comme par exemple basées sur la réflexion et le cryptage bytecode.

4.2.2 SAMADroid

Saba Arshad, Munam A. Shah, Abdul Wahid, Amjad Mehmoud, Hhoubing Song et Hongnian Yu [16], ont proposé SAMADroid : Un nouveau modèle hybride de détection de logiciels malveillants à 3 niveaux pour le système d'exploitation Android. Il s'agit d'un hybride entre trois niveaux d'analyses et de détection de logiciels malveillants : i) Analyse statique et dynamique ; ii) Hôte local et distante ; iii) Intelligence d'apprentissage automatique. En phase d'analyse statique, des expériences sont effectuées pour la sélection des fonctionnalités, afin d'obtenir les fonctionnalités qui peuvent fournir des informations maximales et utiles sur le comportement de l'application. À cet effet, l'ensemble de fonctionnalités de Drebin sont utilisés, avec peu de modifications. Drebin est un cadre de détection de malwares d'analyse statique qui détecte les malwares avec une grande précision que nous avons précédemment présenté, Pour l'analyse dynamique, les appels système sont tracés lors de l'exécution. Différents algorithmes d'apprentissage automatique sont appliqués et leurs performances sont comparées pour obtenir la technique d'apprentissage automatique la plus précise [16].

Fonctionnement de l'outil

SAMADroid est un modèle de détection de malware hybride à 3 niveaux pour les appareils Android. Il est hybride entre les trois niveaux suivants pour l'analyse et la détection des logiciels malveillants. Les trois niveaux du modèle SAMADroid sont illustrés dans la figure 4.2 [16].

1. **Analyse statique et dynamique :** Au niveau 1, l'hybride de l'analyse statique et dynamique fournit une analyse très précise car elle combine les avantages de deux techniques d'analyse. Grâce à l'analyse statique, il scanne tout le code d'application et analyse le comportement malveillant de l'application sans l'exécuter. En phase d'analyse statique, les fonctionnalités statiques sont extraites du fichier manifeste et des fichiers de code dex de l'application. Motivés par Drebin, les mêmes ensembles de caractéristiques statiques sont utilisés pour l'analyse statique, avec une petite altération, afin d'obtenir une précision de détection élevée.

   En phase d'analyse dynamique, le système exécute l'application sur le périphérique réel et analyse son comportement d'exécution, ce qui inclut également la surveillance du code chargé et décrypté dynamiquement. Les appels système sont utilisés comme fonctionnalités dynamiques. Les applications installées sur de vrais appareils Android sont analysées par

le suivi des appels système. Ces appels nous permettent de dépasser les limites de l'analyse statique et d'analyser le comportement de l'application en environnement temps réel.

2. **Hôte locale et distante :** Le niveau 2 est un hybride d'hôte local et distante. Une analyse statique détaillée est effectuée sur l'hôte distante pour obtenir des résultats très précis. Sur l'hôte locale, une analyse dynamique est effectuée pour prendre les entrées réalistes de l'utilisateur au lieu d'utiliser un outil programmé, qui génère des événements d'entrée aléatoires non réalistes. Sur la base des entrées utilisateurs, des journaux d'appels système sont générés et transmis au serveur distant. Le serveur distant continue d'analyser le comportement de l'application sur la base des journaux et des fonctionnalités statiques extraites.
3. **L'intelligence d'apprentissage automatique :** Au niveau 3, les vecteurs de fonctionnalités construits à partir des fonctionnalités analysées sont fournis en entrée à l'unité d'intelligence d'apprentissage automatique pour effectuer la détection des comportements malveillants des applications inconnues et les classer correctement. Toutes les applications sont classées comme malveillantes ou bénignes. Dans SAMADroid, l'opération de détection est effectuée sur l'hôte distante, conservant ainsi tous les ensembles de données d'apprentissage en mémoire du serveur, qui est un système riche en ressources. Cela réduit finalement la surcharge de mémoire.

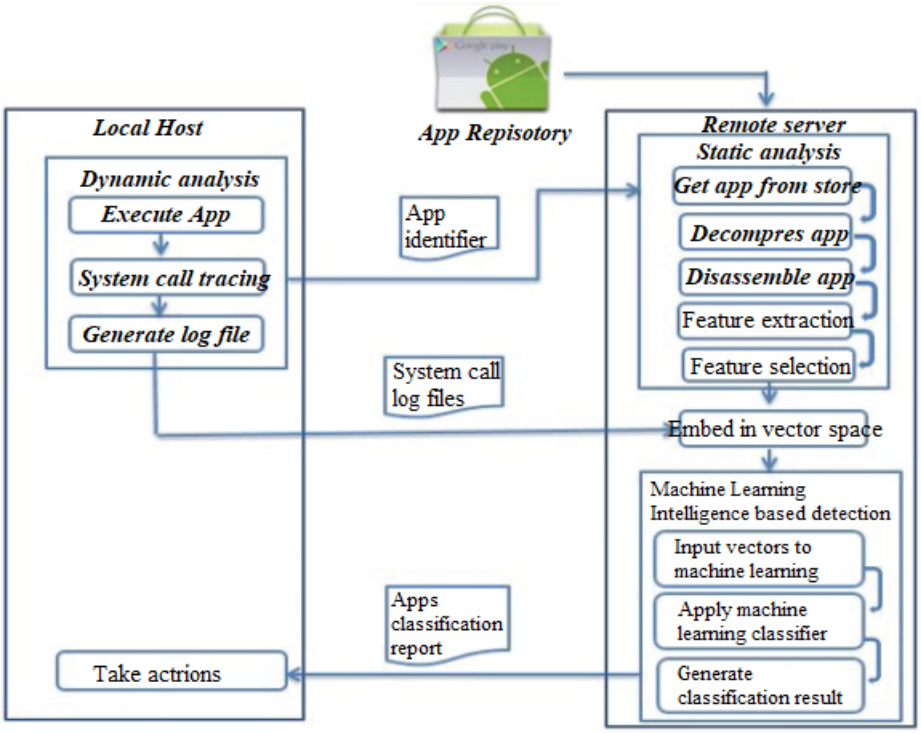


FIGURE 4.2 – Schéma architectural de SAMADroid [16].

Les limitations

— Aucune détection de comportement malveillant n'est effectué sur l'hôte locale, c'est-à-dire l'appareil Android.

— si une congestion se produit au niveau du canal en raison de laquelle l'appareil Android ne peut pas communiquer avec le serveur.

— l'ensemble de données Drebin des applications malveillantes a été utilisé pour former le modèle de classification qui ne contient pas les variantes les plus récentes des types de logiciels malveillants.

4.2.3 AndroPyTool

Martín, Alejandro Lara-Cabrera, Raúl Camacho et David [40], ont proposé, Détection de logiciels malveillants Android par fusion de fonctionnalités hybrides et classificateurs d'ensemble : le cadre AndroPyTool et l'ensemble de données OmniDroid. Il s'agit d'un framework intégré développé dans Python visant à obtenir diverses fonctionnalités dynamiques et statiques à partir d'un ensemble d'applications Android. Il intègre les outils d'analyse des logiciels malveillants Android les plus utilisés, effectue une inspection sur le code source et récupère les informations de comportement lorsque l'échantillon est exécuté dans un environnement contrôlé. L'outil fournit un rapport détaillé pour chaque analyse d'application, y compris un grand lot de grains fins des caractéristiques représentatives.[40][39]

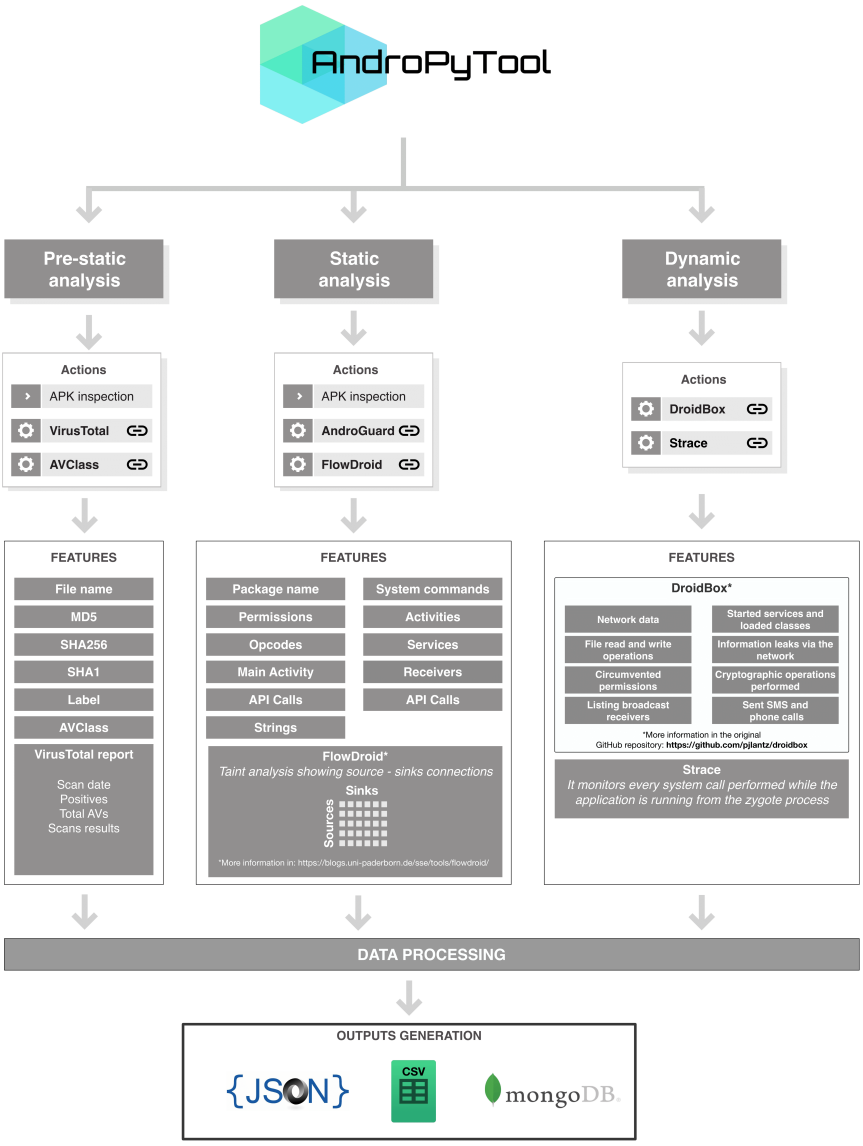


FIGURE 4.3 – Schéma des différentes fonctionnalités et outils d'extraction utilisés dans AndroPyTool [39].

Fonctionnement de l'outil

AndroPyTool est conçu comme un cadre modulaire écrit en Python où un certain nombre de scripts sont exécutés séquentiellement. Cela permet l'intégration en douceur de nouveaux outils d'analyse de logiciels malveillants à l'avenir, ou d'améliorer chacun d'eux séparément. Le fonctionnement d'AndroPyTool suit un processus en sept étapes (voir Figure 4.4) comme décrit ci-dessous :[40]

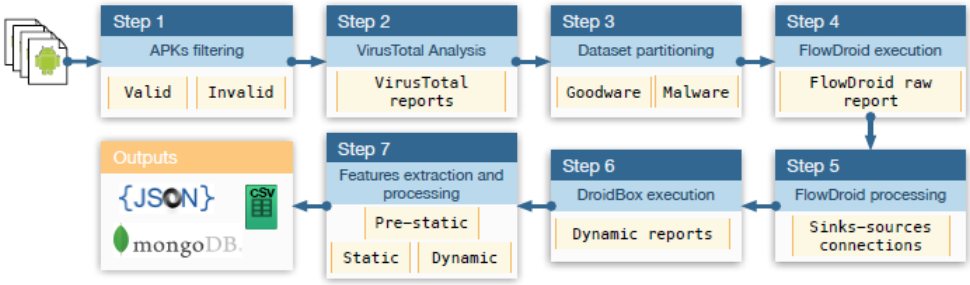


FIGURE 4.4 – Diagramme montrant le processus en sept étapes suivi par AndroPyTool pour extraire un large ensemble de fonctionnalités statiques et dynamiques. [22].

1. **Filtrage APK :** la première étape consiste à inspecter chaque échantillon en utilisant l'outil AndroGuard pour déterminer si l'échantillon est une application Android valide.
2. **Analyse virustotale :** l'outil récupère un rapport de l'application Web en ligne Virustotal. Le rapport contient les résultats du scan effectué par Virustotal ainsi que les résultats donnés de l'analyse de l'application par plus de 60 moteurs anti-malware distincts.
3. **Partitionnement de l'ensemble de données :** chaque échantillon est étiqueté comme un malware s'il est étiqueté de cette manière par au moins un antivirus basé sur Virus-Rapport total. Ce critère peut être modifié par l'utilisateur final de l'outil, qui peut établir son propre seuil ϵ .
4. **Exécution FlowDroid :** cet outil est exécuté sur chaque échantillon.
5. **Traitement des résultats FlowDroid :** il y a un traitement d'étape de ces résultats fournis par FlowDroid, afin d'extraire les liens entre les sources et les puits. La raison de la division de l'extraction et le traitement des flux d'information réside dans le fait que des représentations sont possibles, permettant ainsi de modifier l'étape de traitement indépendamment.
6. **Exécution de DroidBox :** une version personnalisée de DroidBox , un outil d'analyse dynamique Android, qui comprend l'outil Strace est exécuté contre l'échantillon dans cette étape.
7. **Extraction de caractéristiques :** dans cette étape, l'outil rassemble, réorganise et structure les résultats sous forme de rapports d'extraction des caractéristiques. L'ensemble de données combiné des entités extraites pour toutes les applications est fournie dans les formats suivantes : en tant que valeur séparée par des virgules (CSV), en tant qu'objet JavaScript Fichier de notation (JSON), et en tant qu'une base de données MongoDB [1].

1. Un système de gestion de base de données open source (SGBD) qui utilise un modèle de base de données orienté document

Les limitations

— L'approche de fusion d'AndroPyTool est complexe.

— temps de calcul trop lent.

4.2.4 Synthèse

Après avoir étudié les différentes approches DREBIN, SAMADroid et AndroPyTool. Nous avons remarqué que certaines d'elles sont plus performantes que d'autres. Le tableau 4.1 représente une comparaison de ces trois approches que nous avons détaillé.

réf	Approche	Modèle	hybride de	Bénigne	Malware	Accuracy(%)
[15]	DREBIN	SVM	analyse statique	123,453	5,560	93.90.
[16]	SAMADroid	SVM	Analyse statique et dynamique	123,453	5,560	98.75
[40]	ANdroPyTool	Random forest, Bagging classifier	analyse statique et dynamique	11000	89000	89.7

TABLE 4.1 – Comparaison des différents outils de détection de malwares android utilisant l'apprentissage automatique.

4.3 L'apprentissage profond pour la détection des malwares Android

4.3.1 DL-Droid

Alzaylaee, Mohammed K. Yerima, Suleiman Y. Sezer etSakir [13], ont présenté DL-Droid : Détection des malwares Android basée sur l'apprentissage profond à l'aide de vrais appareils. Il s'agit d'un système d'apprentissage profond pour détecter les applications Android malveillantes grâce à une analyse dynamique utilisant une génération d'entrée dynamique. Des expériences réalisées avec plus de 30000 applications (bénignes et malveillantes) sur des appareils réels. DL-Droid utilise une approche de génération d'entrée basée sur l'état pour une couverture de code améliorée permettant ainsi des performances améliorées [13].

Fonctionnement de l'outil

Une plate-forme automatisée est nécessaire pour exécuter les applications Android et extraire leurs fonctionnalités. Ces fonctionnalités seront utilisées comme entrées pour la classification basée sur l'apprentissage profond de DL-Droid afin de détecter les logiciels malveillants Android. Avec l'analyse dynamique des applications Android, la génération d'entrées de test est nécessaire afin d'assurer une couverture de code suffisante pour déclencher les comportements malveillants. La durée exécuté était différente à chaque exécution et déterminé par la génération d'entrée de test choisie. L'horaire requis a été confirmé après évaluation avec plusieurs applications pour déterminer le temps nécessaire pour déclencher chaque événement possible à l'aide de l'outil avec état (DroidBot [2]). Pour la méthode avec état, 180 ont été jugés suffisants. Pour la génération sans état utilisant Monkey, 300 suffisaient pour générer 4000 événements pour les applications. Au-delà de 4000 événements, la plupart des applications n'ont généré aucune sortie dynamique supplémentaire à partir de Dynalog [3]. La vue d'ensemble du processus DL-Droid utilisant DynaLog ainsi que le moteur de classificateur DL est illustrée à la Figure 4.5 [13].

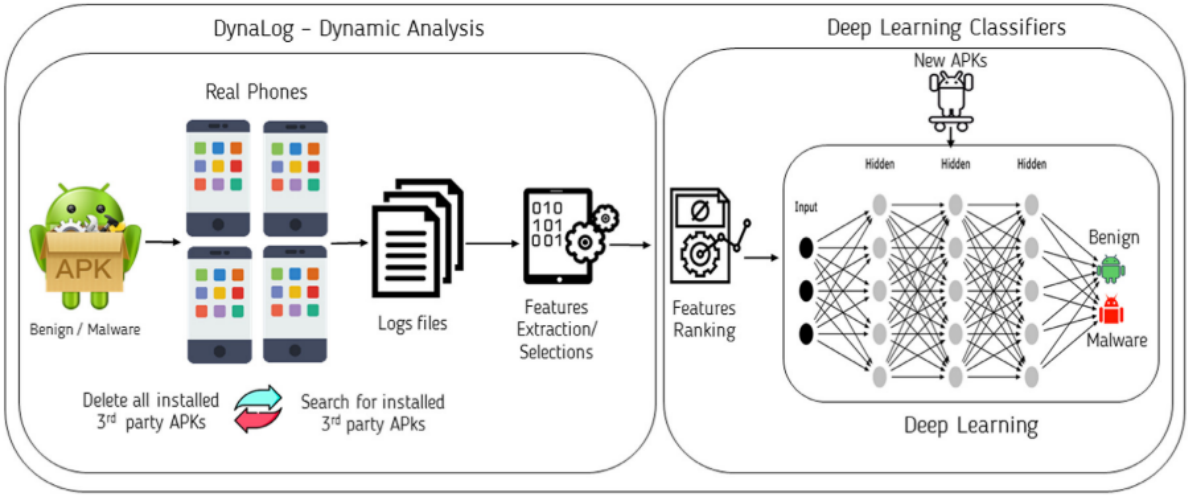


FIGURE 4.5 – L'architecture du DL-Droid [13].

Les limitations

— DL-Droid utilise une large analyse dynamique du code source de chaque échantillon de l'ensemble de données, qui peut consommer beaucoup d'énergie.

— prend beaucoup de temps de calcul.

2. Un générateur d'entrée de test léger et guidé par l'interface utilisateur pour Android
3. Un cadre d'analyse dynamique automatisé pour caractériser les applications Android

4.3.2 MalDozer

ElMouatez Billah Karbab, Mourad Debbabi, Abdelouahid Derhab et Djedjiga Mouheb [19], ont proposé MalDozer : Un cadre automatique pour la détection des logiciels malveillants Android à l'aide de l'apprentissage profond. Il s'agit d'un cadre de détection automatique de malwares Android et d'attribution de famille qui repose sur la classification des séquences à l'aide de techniques d'apprentissage profond. À partir de la séquence brute des appels de méthode API de l'application, MalDozer extrait et apprend automatiquement les modèles malveillants et bénignes des échantillons réels pour détecter les logiciels malveillants Android. Il peut aussi servir de système de détection de malware omniprésent qui est non seulement déployé sur les serveurs, mais aussi sur les appareils mobiles et même IoT(Internet of Things) [19].

Fonctionnement de l'outil

MalDozer a une conception simple, où un prétraitement minimaliste est utilisé pour obtenir les méthodes d'assemblage. Quant à l'extraction des caractéristiques (apprentissage de la représentation) et à la détection / attribution, elles sont basées sur le réseau neuronal réel. Cela permet à MalDozer d'être très efficace avec un prétraitement rapide et une exécution du réseau de neurones. Étant donné que MalDozer est basé sur un apprentissage automatique supervisé, il faut d'abord former le modèle. Ensuite, déployer ce modèle avec une procédure de prétraitement sur les appareils ciblés(Figure 4.6). Le flux de travail de MalDozer commence par extraire les séquences d'appels d'API à partir des packages d'applications Android, dans lesquels le fichier DEX est considéré uniquement [19].

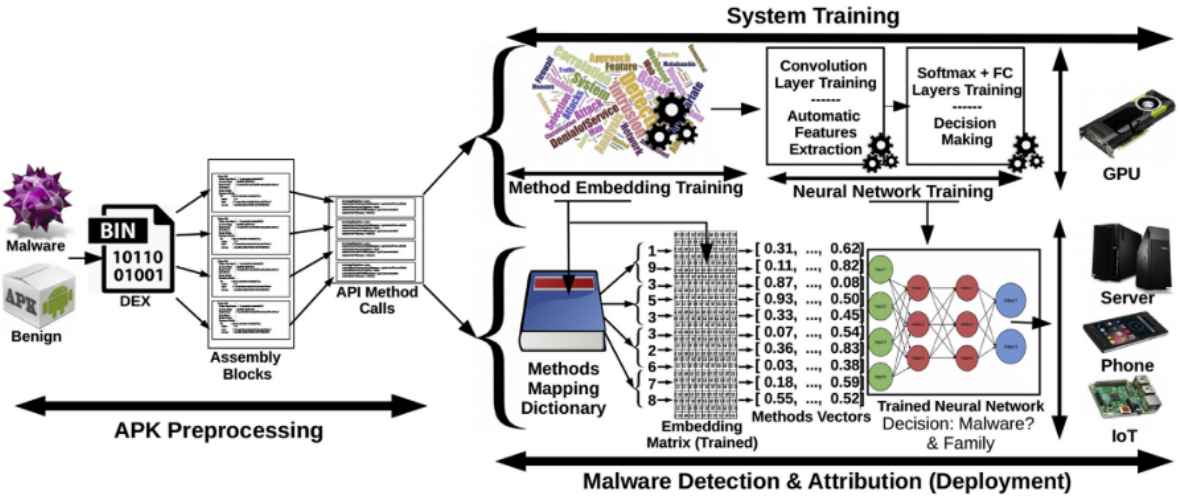


FIGURE 4.6 – L'architecture de MalDozer [19].

Les limitations

— MalDozer extrait seulement les appels de méthode API, qui nécessite a d'autres caractéristiques telle que les permissions, les intents, etc.

— MalDozer n'est pas résistant au chargement du code dynamique.

— MalDozer ne prend pas en compte les codes natifs.

4.3.3 Approche : Utilisation de Réseaux de neurones profonds (DNN) pour la détection de malwares Android

Abdelmonim Naway et Yuancheng LI [29], ont proposé : Utilisation de Réseaux de neurones profonds (DNN) pour la détection de malwares Android. C'est un système de détection de logiciels malveillants Android qui applique une technique d'apprentissage profond pour faire face aux menaces des logiciels malveillants Android. Des expériences approfondies sur un ensemble de données du monde réel contiennent des applications bénignes et malveillantes [29].

Fonctionnement de l'outil

La figure 4.7 écrit la conception structurelle de la méthodologie proposée par les auteurs de cet article. La méthodologie proposée comprend les étapes suivantes [29] :

1. **Décompilation des applications :**Le cadre de sécurité mobile qui, est un outil puissant pour l'analyse des logiciels malveillants Android qui est capable de récupérer le code des applications obscurcies a été utilisé pour décompresser et décompiler les fichiers APK.
2. **Extraction des fonctionnalités :** MobSF est également utilisé pour extraire les fonctionnalités de deux fichiers, d'abord, Androidmanifest.xml, qui incluent les filtres Autorisations et Intent-filter. Ensuite, les appels des API sont tirés des fichiers smali générés par MobSF. Le dossier des ressources est filtré pour la présence de fichiers APK. Enfin, la légitimité du certificat est vérifié.
3. **Vecteur d'entrer :** Toutes les entités extraites sont transformées en un vecteur d'entrer.
4. **Classificateur DNN :** En s'appuyant sur les fonctionnalités extraites que DNN étudie pour caractériser les applications Android.
5. **Classification :** En appliquant le classificateur DNN, les nouvelles applications inconnues seront étiquetées comme bénignes ou malveillantes.

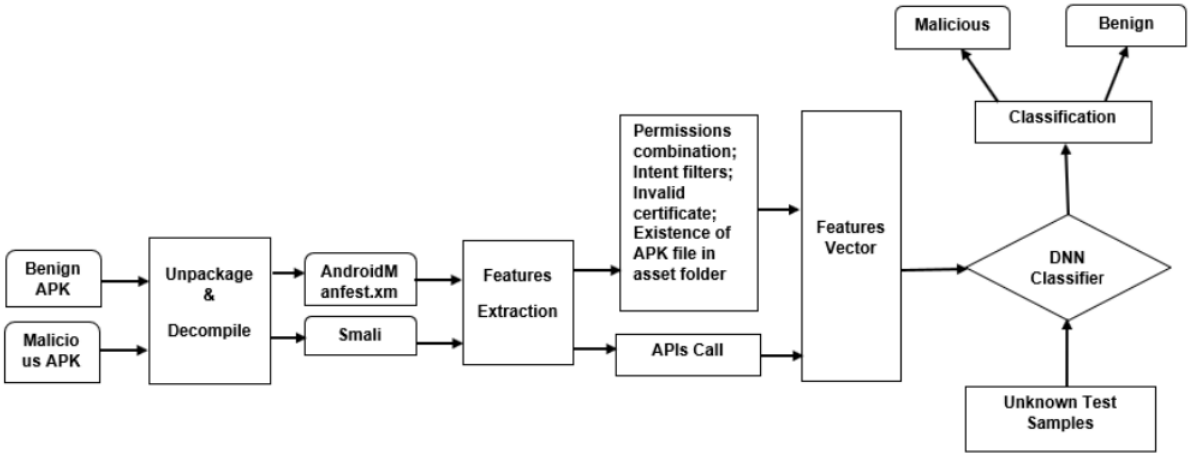


FIGURE 4.7 – L'architecture de l'outil [29].

Les limitations

— L'ensemble de données utilisé par cette approche ne contient pas les variantes les plus récentes des types de logiciels malveillants.

4.3.4 Synthèse

Après avoir étudié les différentes approches DL-Droid, Maldozer et l'outil de détection de malwares utilisant les DNN. Nous avons remarqué que certaines d'elles sont plus performantes que d'autres. Le tableau 4.2 représente une comparaison de ces trois approches que nous avons détaillé.

réf	Approche	modele	Hybride de	bénigne	malware	Nombre de neurones	Accuracy(%)
[13]	DL-Droid	MLP	analyse dynamique	19,620	11,505	[300,100,300]	94.95
[19]	Maldozer	CNN	analyse statique	37,627	20,089	N/A	96.29
[29]	Approche 3	DNN	analyse statique et dynamique	600	600	[200,150,100]	95.31

TABLE 4.2 – Comparaison des différents approches de détection de malwares utilisant l'apprentissage profond.

4.4 Synthèse

Les approches étudiées d'apprentissage automatique et profond pour la détection des malwares Android décrivent tous les deux des méthodes d'apprentissage pour apprendre et prendre des décisions. L'apprentissage profond est une branche de l'apprentissage automatique, et les effets des modèles du DL sont évidemment supérieurs à ceux des méthodes traditionnelles du ML (ou modèle peu profond) dans la plupart des scénarios d'application.

Nous avons proposé une comparaison basée sur les critères suivants : le taux d'exactitude, analyse hybride, temps minimisé, efficacité. Le tableau 4.3 résume cette comparaison entre les travaux étudiés dans ce chapitre ; le signe **(√)** signifie que l'approche a atteint le critère , contrairement au signe **(X)**, le signe **(-)** signifie que l'approche a atteint le critère mais pas d'une manière permanente.

réf	Approche	Taux d'exactitude	analyse hybride	temps minimisé	efficacité
[15]	DREBIN	√	X	X	-
[16]	SAMADroid	√	√	X	-
[40]	ANdroPyTool	√	√	X	√
[13]	DL-Droid	√	X	X	√
[19]	Maldozer	√	X	√	√
[29]	Approche 3	√	√	X	-

Table 4.3 – Comparaison des approches étudiées.

Nous avons conclu de l'analyse d'études précédentes que :

- Les travaux étudiés ont prouvé l'intérêt d'effectuer une analyse hybride pour l'ensemble de données collecté, et prendre en considération le temps de calcul, qui est un facteur important.
- L'apprentissage profond est traditionnellement utilisé pour de très grandes caractéristiques, de sorte que les modèles peuvent être formés pour prendre de nombreuses décisions en couches. L'apprentissage automatique classique utilise un ensemble de données qui contient moins de caractéristiques.
- Bien que l'apprentissage profond puisse bien apprendre sur de nombreuses données, il existe de nombreux problèmes où il n'y a pas suffisamment de données disponibles pour qu'il soit utile. L'apprentissage profond et l'apprentissage automatique partagent des limites statistiques standard et peuvent être biaisés s'il a été collecté avec des techniques statistiques inappropriées.

4.5 Conclusion

Après avoir étudié les différents travaux de l'apprentissage automatique et l'apprentissage profond et leurs utilisations dans la détection des malwares Android, dans le chapitre suivant nous allons présenter notre contribution où nous utilisons l'une des techniques d'apprentissage profond.

CHAPITRE 5

CONTRIBUTION ET IMPLÉMENTATION

5.1 Introduction

Après avoir étudié l'état de l'art des différents outils d'apprentissage automatique et profond pour les systèmes de détection de malwares Android, nous présentons notre contribution, en commençant par l'architecture de notre système, dans cette partie nous détaillons l'architecture proposée. Ensuite nous définissons l'architecture de notre modèle. Enfin nous conclurons avec l'implémentation en mettant l'accent sur l'ensemble de données utilisé, les résultats du modèle proposé et les interfaces de notre application.

5.2 Architecture proposé

Notre projet porte sur la tâche d'analyse des applications Android et la détection des applications malveillantes. Pour atteindre cet objectif et pour obtenir la meilleure performance possible, nous proposons le système suivant :

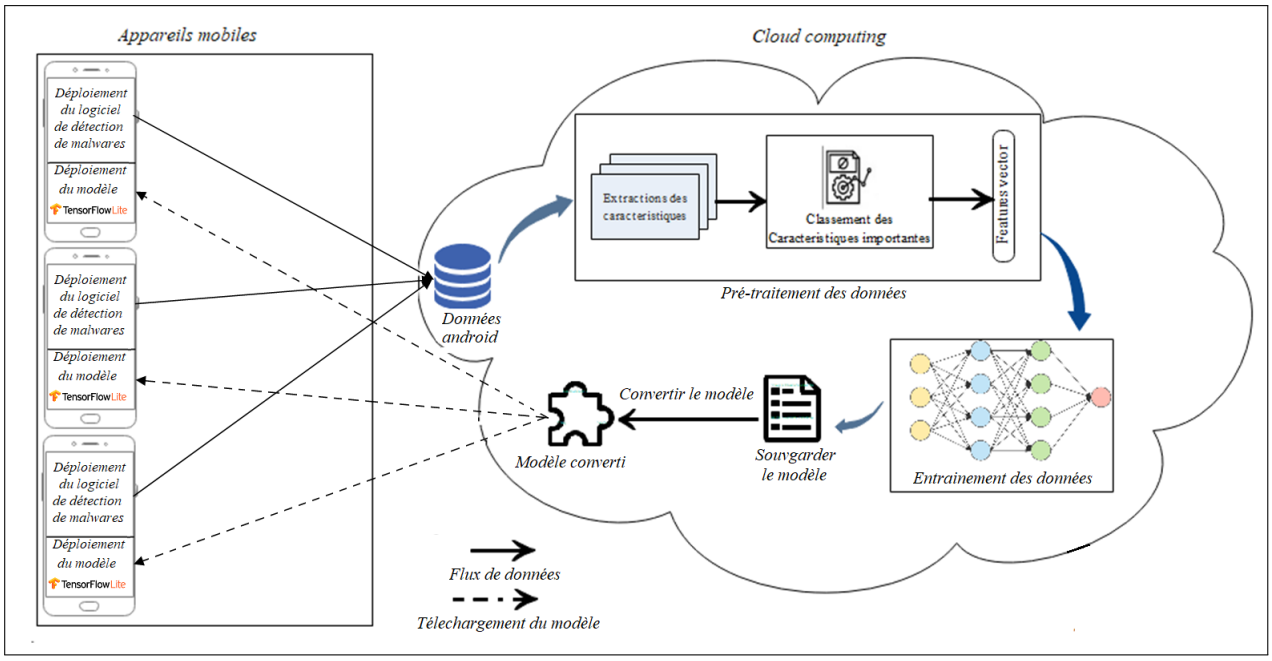


FIGURE 5.1 – L'architecture du système.

Le système est composé de deux couches, couche des appareils mobiles et la couche du cloud computing, où chaque couche encapsule plusieurs étapes qui existe successive. Nous détaillons dans ce qui suit toutes les étapes.

5.2.1 La couche des appareils mobiles

Cette partie contient plusieurs appareils mobiles différentes, chaque appareil a deux taches :

1. **Déploiement du logiciel de détection de malwares :** La première étape est l'installation de l'application au niveau de l'appareil mobile, ensuite l'application envoie les données concernant les applications des mobiles au cloud computing.
2. **Déploiement du modèle :** Cette partie est la dernière tache où le modèle de détection sera téléchargé dans l'appareil mobile pour être déployer et utiliser dans l'application de détection des logiciels malveillants.

5.2.2 Couche cloud computing

Cette couche suit les étapes suivantes :

1. **Extraction de l'ensemble de données Android :** Le processus d'exploration de données (KDD) est appliqué pour l'ensemble de données collectés depuis les appareils mobiles qui comprend la préparation,la sélection de données et le nettoyage des données sur cet ensemble de données.

   Après, on passe à l'étape de l'étiquetage de ces données comme étant Bénignes ou malveillantes, cet étiquetage se fait à l'aide du service d'analyse d'application :**VirusTotal**.

Toutes les applications dans le cloud doivent être scannés avec plus de 70 scanners antivirus, où chaque application sera étiqueté comme malware ou non. Chaque échanillon est étiqueté comme malware s'il est marqué par au moins d'un antivirus, où 1 indique que l'application est malveillante et 0 non malveillante.

2. **Pré-traitement des données :** Cette partie a plusieurs étapes à suivre, nous les détaillons ci-dessus :

   a) **Extraction des caractéristiques :** Une caractéristique est une propriété ou caractéristique estimable importante d'un événement sous observation. Chaque attribut de l'ensemble de données signifie une caractéristique unique et chaque entité signifie un échantillon de données (un APK dans ce cas). Une caractéristique correspond à une autorisation spécifiée par une application.

   Nous avons utilisé un outil d'analyse statique AndroGuard pour extraire les caractéristiques suivantes : (a) Permissions, (b) Appels API, et l'outil d'analyse dynamique DroidBox pour extraire les caractéristiques suivantes : (c) Commandes liées à l'APK. Nous avons extrait toutes les caractéristiques de chaque échantillon, où nous avons crées un vecteur binaire pour chacun. le vecteur d'un échantillon est représenté de la manière suivante, où (F_1) : permissions de l'apk, (F_2) : Appels API et (F_3) : Commandes liées.

$$S_i = \{APK, F_1, F_2, F_3, Type\} \tag{5.1}$$

   Le tableau 5.1 présente un aperçu des caractéristiques de leurs différentes catégories.

Type	caractéristiques
Appels API	abortBroadcast, getDeviceId, getSubscriberId, getCallState, getSimSerialNumber, android.provider.Contacts, android.provider.ContactsContract, getNetworkOperator, getSimOperator
Commande liées	.apk ; pmsetInstallLocation ; pminstall ; GET-METADATA ; GET-RECEIVERS ; GET-SERVICES ; GET-SIGNATURES ; GET-PERMISSIONS
Permissions	android.permission.UPDATE-LOCK ; android.permission.VIBRATE ; android.permission.WAKE-LOCK ; android.permission.WRITE-CALENDAR ; android.permission.WRITE-CONTACTS ; android.permission.ACCESS-FINE-LOCATION

TABLE 5.1 – Aperçu des fonctionnalités extraites des applications [51].

L'ensemble de données combine les caractéristiques extraites pour toutes les applications qui est fourni dans le format (CSV).

b) **Classement des caractéristiques importantes :** Les ensembles de données énormes et complexes affectent les modèles de formation en réduisant l'efficacité du modèle en utilisant un nombre inutile de caractéristiques, ce qui augmente le temps de calcul. Par conséquent, l'extraction de caractéristiques importantes est un élément clé avant la modélisation. Après avoir formé l'ensemble de données qui contient toutes les caractéristiques de tous les échantillons sous forme binaire, nous classons toutes ces caractéristiques en les minimisant et en supprimant celles qui sont redondantes. Il existe une méthode pour traiter ce problème : ExtraTreesClassifier, qui réduit l'ensemble de caractéristiques à un ensemble plus petit, après nous formons un autre ensemble de données qui contient que les caractéristiques importantes. La figure suivante représente un aperçu des caractéristiques importantes extraites de nos données où (x_i) est une caractéristique, toutes les caractéristiques importantes représentent un vecteur de caractéristiques :

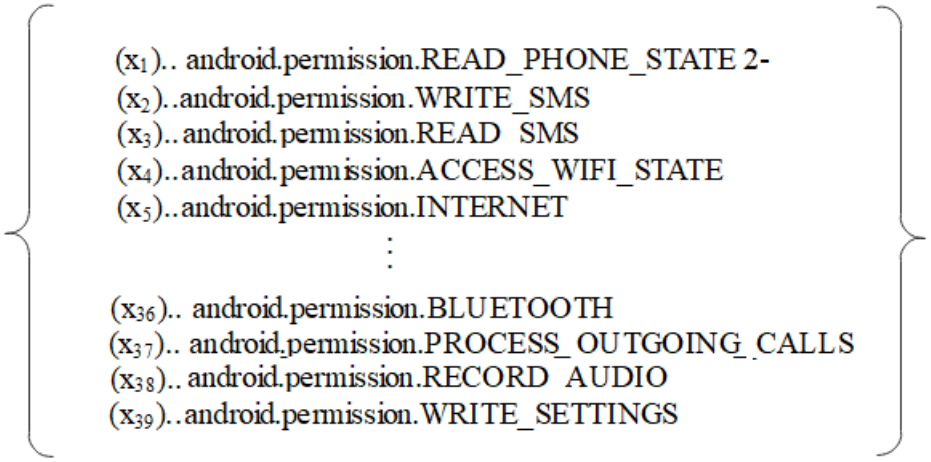


FIGURE 5.2 – Extrait du vecteur de caractéristiques de notre projet.

c) **Vecteur d'entrer :** Toutes ces caractéristiques importantes extraites sont transformées en un vecteur d'entrer. Ce vecteur contient des données sous forme binaire où «1» indique la présence d'une caractéristique importante et «0» indique l'absence.

3. **Entrainement des données :** En s'appuyant sur le vecteur d'entrer, nous utilisons les différents classificateurs d'apprentissage automatique et profond pour définir le meilleur classificateur pour les données utilisées.
4. **Sauvegarder le modèle :** Après la création du modèle, nous le sauvegardons d'abord

dans le cloud, après nous le convertirons pour le déployer et l'utiliser dans les appareils mobiles Android.

5.3 Modèle de détection proposé

Comme moteur de notre modèle de détection, nous avons utilisé l'apprentissage profond vue le nombre élevé de caractéristiques qui nécessites un modèle d'apprentissage profond qualifié. Un autre facteur important à tenir en compte est que cet ensemble de données nécessite plusieurs couches cachées pour obtenir les meilleurs résultats possibles. Le problème avec les algorithmes d'apprentissage automatique est qu'ils pourraient ne pas être en mesures d'apprendre la non-linéarité complexe qui pourrait se trouver dans les données utilisées.

Au cours de nos expériences, nous avons effectués plusieurs test des différentes méthodes d'apprentissage profond, où nous avons testé plusieurs combinaison de couches et au nombre de neurones, fonction d'activation, la taille de lot [1](batch size) et aussi le nombre d'époques [2](epochs), afin d'améliorer les performances du modèle que ce soit en temps ou en efficacité. La figure suivante représente l'architecture de notre modèle :

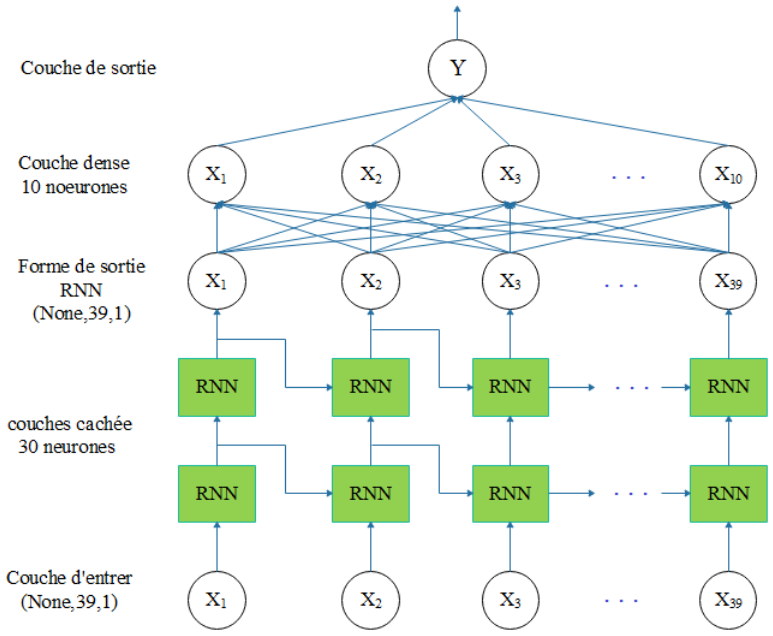


FIGURE 5.3 – Architecture générale du modèle RNN.

Chaque modèle a une couche d'entrer, des couches cachées et une couche de sortie, ces couches sont détaillé comme suit :

1. Définit le nombre d'échantillons qui seront propagés à travers le réseau.
2. Le nombre de passages complets dans l'ensemble de données de formation

- **La couche d'entrer :** Contient les données extraites du vecteur de caractéristiques avec le total des échantillons, le vecteur est représenté en binaire, voici un example de données d'entrer :

  X = [0,0,0,0,0,0,0,1,0,1,0,0,0,0,0,0,0,1,0,1, 0,0,0,0,0,0,0,1,0,1,0,0,0,0,0,0,0,1,0]

- **Les couches cachées :** Nous avons utilisé plusieurs couches cachées, RNN, Dropout, Norme par lots et Dense :
  - **La couche RNN :** Nous avons utilisé deux couches RNN de taille 30 suivi d'une fonction d'activation relu.
  - **La couche norme par lots(Batch Normalization) :** Pour améliorer la vitesse, les performances et la stabilité des RNN, nous avons ajouté 2 couche Batch normalization.
  - **La couche Dropout :** Nous avons appliqué deux couches Dropout après les deux couches RNN pour la régularisation.
  - **La couche Dense(ou entièrement connecté) :** Nous avons ajouté une couche Dense, qui contient 10 neurones suivi d'une couche d'activation relu.
- **La couche de sortie :**Nous avons ajouté une autre couche dense celle de la sortie, elle contient un seul neurone pour prédire la sortie suivi d'une fonction d'activation sigmoïde.

Pour la procédure d'apprentissage finale du modèle, une taille de lot(batch size) de 20 a été fixée et 250 époques(epochs), avec l'optimiseur [3] «Adam» a été utilisé. Enfin, nous sauvegardons les poids et le modèle formé et le vecteur de caractéristiques pour l'utiliser afin de classifier une application inconnue comme étant malwares ou non.

5.4 Implémentation

Cette section décrit les résultats expérimentaux et les expériences réalisées de notre système pour la détection des logiciels malveillants. Vue le manque de moyens et la non disponibilité du cloud computing et d'un nombre important d'appareils mobiles, nous avons utilisé l'ensemble de données décrit ci-dessus, qui contient toutes les caractéristiques que nous avons détaillée dans l'architecture du système proposé.

5.4.1 Ensemble de données

L'ensemble de données utilisé pour l'analyse est fourni par Kaggle [4]. Il a été crée par Christian Urcuqui[49], où il a collecté les données Android malveillantes depuis le projet Android Malware GENOME et les autres données des différents magasins mobiles comme Google PlayStore, toutes ces données collectées ont été analysé par l'outil AndroGuard, où il a extrait trois caractéristiques

3. Permet de réduire le poids des erreurs

4. Une plateforme compétitive où les mineurs de données et les scientifiques peuvent obtenir de nombreux ensembles de données

importantes(Permissions,Appels API et les commandes liées). Les appels d'API sont des appels qui sont effectués vers le serveur au nom d'une application à l'aide d'un SDK [5] ou d'une API. Les permissions Android sont des demandes ou des autorisations acquises par les applications afin d'utiliser certaines données et fonctionnalités du système pour maintenir la sécurité du système et de l'utilisateur. Les commandes liées sont les commandes standard de l'OS et du framework qui s'activent lors de l'exécution du code source de l'application[47]. Après il a crée un vecteur binaire de fonctionnalités utilisé pour chaque application analysée 1 = utilisé, 0 = non utilisé. De plus, les échantillons de logiciels malveillants / bénignes ont été divisés par "Type" ; 1 malware et 0 non-malware. Cette ensemble contient les fonctionnalités de 398 applications qui sont étiquetées comme «bénignes» ou «malveillantes». Cette étiquette est utilisée pour effectuer la sélection des fonctionnalités et l'analyse de la classification par apprentissage supervisé. L'ensemble de données comprend un fichier .csv. La figure suivante représente un extrait de cet ensemble de données :

	android	android.app.cts.permission.TEST_GRANTED	android.intent.category.MASTER_CLEAR.permission.C2D_MESSAGE	android.o
0	0	0	0	
1	0	0	0	
2	0	0	0	
3	0	0	0	
4	0	0	0	
...	...	...	...	
393	0	0	0	
394	0	0	0	
395	0	0	0	
396	0	0	0	
397	0	0	0	

398 rows × 331 columns

FIGURE 5.4 – Extrait de l'ensemble de données utilisé.

5.4.2 Résultats expérimentaux

Dans cette section, nous adaptons notre modèle proposé RNN. Ensuite, nous comparons les résultats obtenus à certains classificateurs d'apprentissage automatique et d'apprentissage profond. Dans cette étude comparative, nous avons utilisé comme ensemble de données de d'apprentissage 80% de la totalité des données et comme données de test 20% de l'ensemble de données.

Évaluation de notre modèle

L'évaluation de notre modèle se base sur quatre mesures : vrai positif, vrai négatif, faux positif et faux négatif. Ces valeurs sont utilisées pour calculer les mesures de performance pour le modèle

5. (kit de développement logiciel) est un ensemble d'outils de développement utilisés pour développer des applications pour la plateforme Android.

proposé. Nous considérons cinq paramètres pour l'évaluation :F-mesure, Taux d'exactitude(ACC), précision(P), taux de détection(DR), Taux de fausses alarmes(FAR).

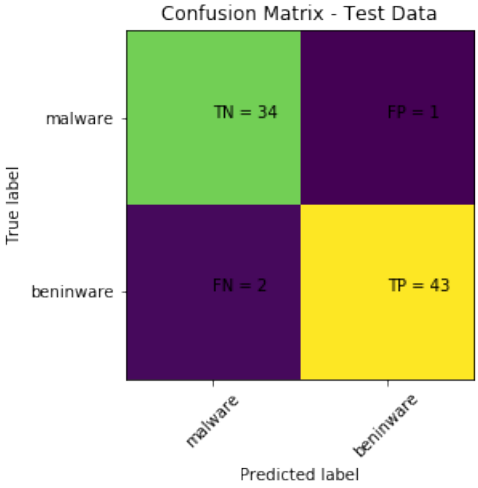


FIGURE 5.5 – Matrice de confusion de notre modèle.

Ces mesures de performance sont calculé à travers les résultats de la matrice de confusion démontré dans la figure 5.5, par exemple le taux de détection(DR) est calculé comme démonte l'équation ci-dessus, la même procédure a été effectuée pour toutes les autres mesures.

$$DR = \frac{TP}{TP + FN} = \frac{43}{43 + 2} = 0.9555555555556 \tag{5.2}$$

	ACC	P	DR	FAR	F-mesure
RNN	**99,37%**	**97,72%**	**95,56%**	**2,86%**	**96,63%**

TABLE 5.2 – Les mesures d'estimation de notre modèle.

Le tableau 5.2 illustre que le taux d'exactitude du système proposé à identifier une application comme bénigne ou malveillante est de 99,37%. La précision des applications correctement classées est de 97,72%. Le taux de détection des échantillons malveillants est de 95,56%. La mesure-F qui indique à quel point le modèle déterminé est de 96,63%.

Étude comparative

Des séries d'expériences ont été réalisées lors du développement de notre modèle, où nous avons vérifié les performances sur les données de test. Pour analyser les performances de notre modèle, nous l'avons comparé avec différents modèles d'apprentissage automatique et profond comme le montre le tableau 5.3.

	ACC	P	DR	FAR	F-mesure
CNN1D	98,44%	95,35%	88,57%	8,89%	93,18%
CNN2D	98,56%	91,30%	91,18%	8,70%	92,31%
DNN	99,37%	95,45%	91,67%	4,55%	94,38%
CNN-RNN	99,06%	89,13%	88,23%	10,87%	90,11%
RNN	**99,37%**	**97,72%**	**95,56%**	**2,86%**	**96,63%**
SVM	95,91%	93,33%	91,43%	6,67%	93,33%
Naives bayes	90,25%	92,68%	82,05%	7%	88,37%
KNN	93,08%	90,91%	86,11%	9,09%	89,89%
Desion Tree	92,14%	88,89%	85,71%	11,11%	88,89%
Logistic Regression	93,71%	91,11%	88,57%	8,89%	91,11%

TABLE 5.3 – Comparaison des différentes méthodes d'apprentissage profond avec notre modèle.

On remarque que notre modèle (RNN) donne un des bons résultats comparé aux autres méthodes et on trouve que les méthodes d'apprentissage profond sont plus performantes à celles des autres méthodes d'apprentissage automatique, vue que le problème avec ces méthodes qu'elles ne pourraient pas apprendre la non-linéarité complexe qui pourrait se trouver dans les données.

Les deux figures 5.6 et 5.7 nous montre que notre modèle donne le taux de détection le plus élevé avec 95,56%, la précision la plus élevée avec 99,37% et un FAR faible avec 2,86%.

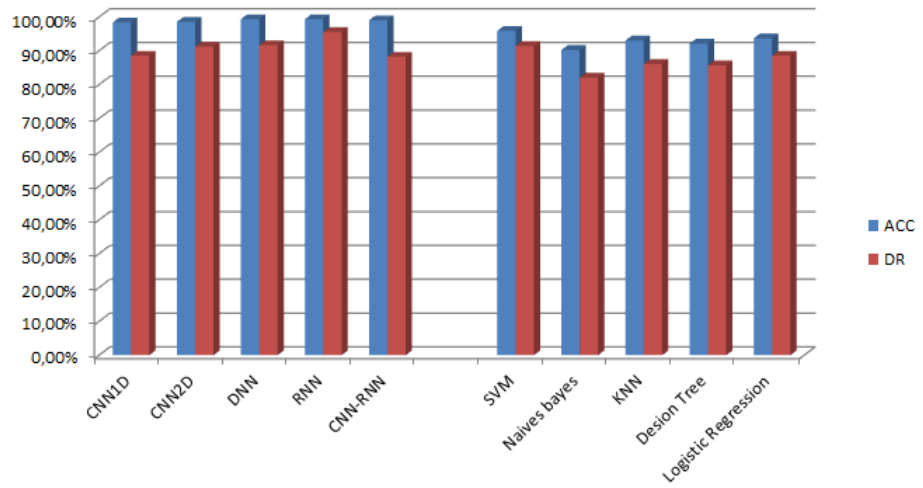


FIGURE 5.6 – Le taux de détection(DR), la précision(ACC) de notre modèle comparé à d'autre modèles d'apprentissage automatiques et profond.

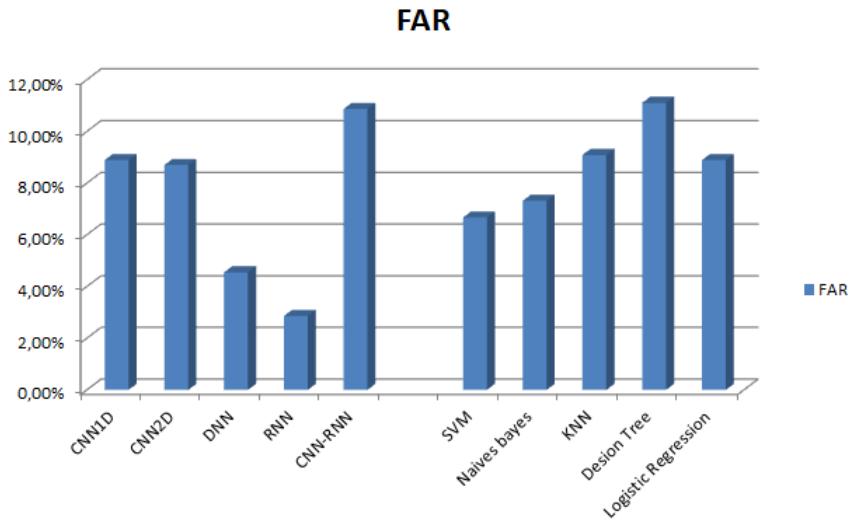


FIGURE 5.7 – Taux de fausses alarme(FAR) de notre modèle comparé à d'autre modèles d'apprentissage automatiques et profond.

5.4.3 Environnement de développement

Environnement et technologies logicielles

1. **Ressources matérielles**

CPU	Intel(R) Core(TM) i7-7500U CPU @ 2.70GHz
GPU	AMD Radeon(TM) R7 M440
RAM	8.00Go

TABLE 5.4 – Ressources matérielles

2. **Ressources logicielles**

SE	Windows 10 Professionnel
HARDWARE	Jupyter NoteBook, Colab, Android Studio

TABLE 5.5 – Ressources logicielles

Outils utilisé

- **TensorFlow**
  TensorFlow est une bibliothèque d'apprentissage automatique, il s'agit d'une boîte à outils permettant de résoudre des problèmes mathématiques complexes plus facilement. Elle permet de développer des architectures d'apprentissage expérimentales et de les transformer en logiciels[33].

TensorFlow regroupe un grand nombre de modèles et d'algorithmes d'apprentissage automatique et profond. Son API front-end de développement d'applications repose sur le langage Python, tandis que l'exécution de ces applications s'effectue en C++ haute-performance. Cette bibliothèque permet notamment d'entraîner et d'exécuter des réseaux de neurones pour la classification d'images, la la détection des malwares, classification des objets, etc.[33].

- **TensorFlow Lite**
  TensorFlow Lite est un ensemble d'outils pour aider les développeurs a exécuter des modèles sur des appareils mobiles, embarqués et IoT. Il permet l'inférence d'apprentissage automatique sur l'appareil avec une faible latence et une petite taille binaire [12].
  TensorFlow Lite se compose de deux composants principaux [12] :
  — **L'interpréteur TensorFlow Lite :** Qui exécute des modèles spécialement optimisés sur de nombreux types de matériel différents, y compris les appareils mobiles, les périphériques Linux intégrés et les micro-contrôleurs.
  — **Le convertisseur TensorFlow Lite :** Qui convertit les modèles TensorFlow en une forme efficace a utiliser par l'interpréteur, et peut introduire des optimisations pour améliorer la taille et les performances binaires.
- **Keras**
  Keras est une bibliothèque open-source de composants de réseau de neurones écrite en Python et capable de fonctionner au-dessus de tensorflow , CNTK ou Théano. Il a été développé pour but de lui permettre une expérimentation rapide [5].
  Utilisation de Keras dans le besoin d'une bibliothèque d'apprentissage profond qui [5] :
  — Permet un prototypage facile et rapide (grâce à la convivialité, la modularité et l'extensibilité).
  — Prend en charge les modèles d'apprentissage profond récents.
- **Scikit-learn**
  Scikit-learn est une bibliothèque en Python qui fournit de nombreux algorithmes d'apprentissage non supervisés et supervisés. Il s'appuie sur certaines des technologies, comme NumPy, pandas et Matplotlib. Les fonctionnalités fournies par scikit-learn incluent [11] :
  — **Régression**, y compris régression linéaire et logistique.
  — **Classification**, y compris K-voisins les plus proches.
  — **Clustering**,y compris K-Means et K-Means ++.
  — **Sélection du modèle**.
  — **Prétraitement**, y compris la normalisation Min-Max.

Autres outils

- **Matplotlib [9] :** est une bibliothèque complète pour créer des visualisations statiques, animées et interactives en Python.

- **pandas [10] :** est une bibliothèque open source qui fournit des structures de données hautes performances et faciles a utiliser et des outils d'analyse de données pour le langage de programmation Python.
- **Numpy [8] :** est une bibliothèque en Python qui fournit de nombreux algorithmes d'apprentissage non supervisés et supervisés. Il s'appuie sur certaines des technologies, comme NumPy, pandas et Matplotlib.

5.4.4 Présentation des Interfaces de l'application "MalDetector"

Cette sous-section présente les différentes interfaces de notre application.

Page d'accueil

Après avoir lancé l'application nommée **"MalDetector"**, l'interface d'accueil suivante est affichée

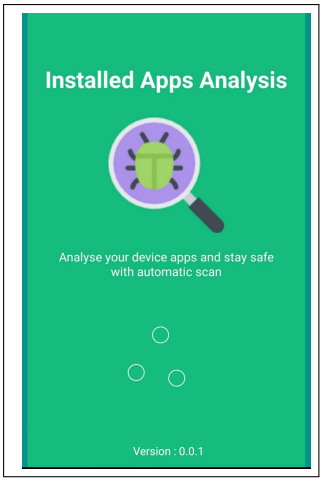


FIGURE 5.8 – Interface de lancement de l'application.

Choix de l'application à scanner

L'application importe toutes les applications installées dans le l'appareil mobile de l'utilisateur, et affiche le nombre total d'applications installées. Toutes ces applications sont représenté sous forme d'une liste(Figure 5.9).

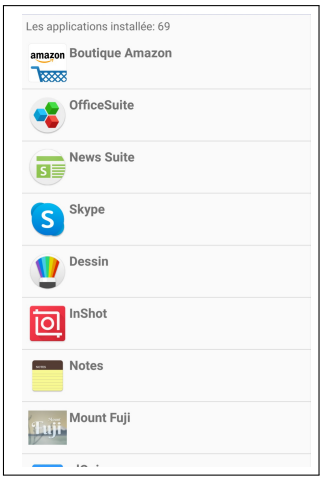


FIGURE 5.9 – Liste d'application installées dans l'appareil mobile de l'utilisateur.

L'utilisateur choisi une application a scanner, une autre interface s'affiche à l'utilisateur, comme le démontre la figure 5.10. L'interface contient deux boutons où il peut faire deux chois, il scanne l'application ou il fait un WI-FI scan(Fonctionnalité supplémentaire).

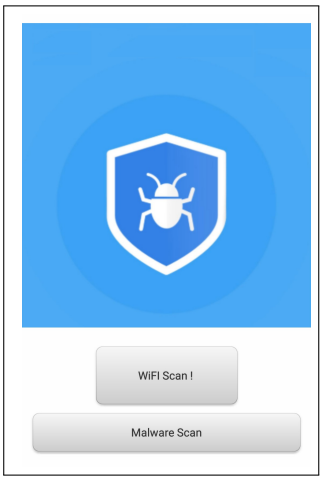


FIGURE 5.10 – Interface du choix d'opération a effectuer.

Scanner l'application

Cette étape contient deux parties, MalDetector prend l'application et extrait les fonctionnalités(Permissions, appels API, commandes liées) et les forme dans un vecteur, ce vecteur sera comparé au vecteur de caractéristiques importantes pour créer le vecteur d'entrer sous forme binaire, où la présence d'une fonctionnalité est indiqué par **1** et l'absence par **0**.

Après avoir formé le vecteur d'entrer, **MalDetector** prend le modèle (modele.tflite) et lance la prédiction, où une interface d'attente de la prédiction d'affiche comme démontre la figure 5.11

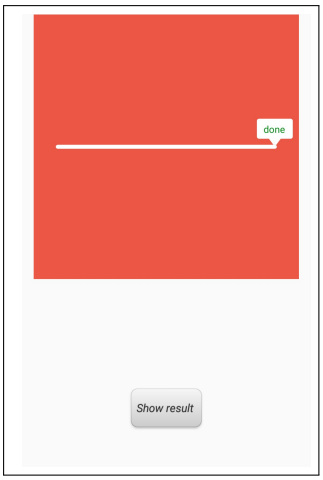


FIGURE 5.11 – Interface d'attente du résultat.

Deux résultats possibles s'affiche à l'utilisateur :

— Soit l'application ne contient aucune caractéristiques malveillantes (le cas d'une application bénigne), l'interface suivante s'affiche à l'utilisateur, le résultat est représenté en pourcentage :

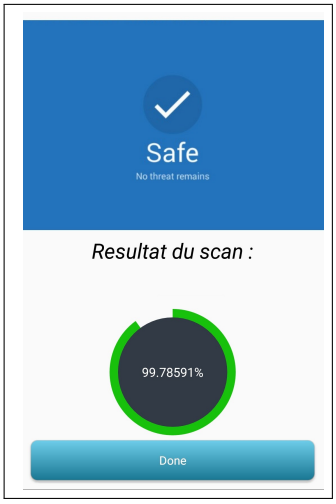


FIGURE 5.12 – Application bénigne.

— Soit l'application contient des caractéristiques malveillantes (le cas d'une application malware), l'interface suivante s'affiche à l'utilisateur, le résultat est représenté en pourcentage :

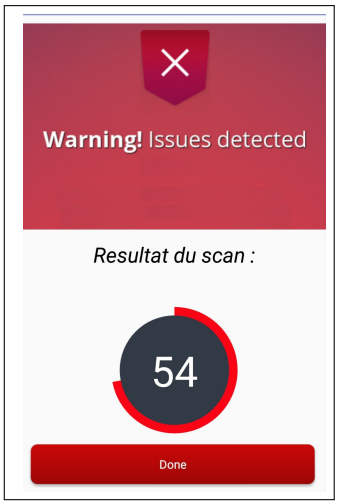


FIGURE 5.13 – Application malware.

WI-FI scan

Nous avons ajouté une fonctionnalité supplémentaire à **MalDetector** qui offre à l'utilisateur les capacités de numérisation Wi-Fi fournies par l' API WifiManager [6] pour obtenir une liste des points d'accès Wi-Fi visibles depuis l'appareil.

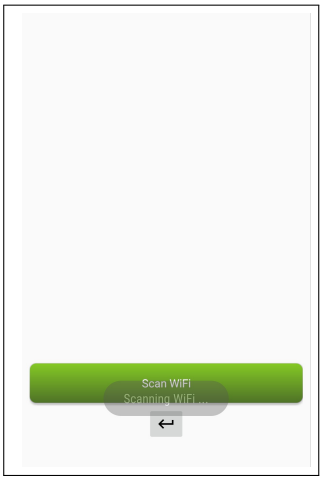


FIGURE 5.14 – WI-FI scan.

5.5 Conclusion

Ce présent chapitre a été consacré aux différents outils ayant contribué à l'aboutissement de ce projet. On a détaillé le système proposé et son fonctionnement. Nous avons aussi présenté l'architecture de notre modèle et l'implémentation, où nous avons décrit l'ensemble de données utilisé, les résultats expérimentaux et les interfaces de notre application.

6. Classe fournit l'API principale pour gérer tous les aspects de la connectivité Wi-Fi

CONCLUSION GÉNÉRALE ET PERSPECTIVES

Dans ce mémoire nous avons proposé et évalué notre nouvelle méthode afin de caractériser, classifier et détecter les malwares Android. Notre approche est classée parmi les techniques d'analyse statique et dynamique. Les logiciels malveillants peuvent être donc identifier automatiquement en utilisant les méthodes d'apprentissage profond. Ce modèle de détection a été nommé **"MalDetector"**.

Les principales contributions et conclusions de ce travail sont les suivantes : Dans cette étude, nous avons utilisé des échantillons de 398 applications récentes malveillantes. Les algorithmes d'apprentissage automatique et profond les plus connus sont utilisés et étudiés.

Le produit final résultant de cette étude est un modèle pour la détection d'applications malveillantes d'Android qui se constitue : D'une procédure d'extraction de caractéristiques à partir d'applications basées sur l'analyse statique du fichier ManifestAndroid et sur l'analyse dynamique du code source. Un réseaux de neurones récurrents(RNN), qui a été configuré et adapté pour être applicable. Notre modèle possède un taux d'exactitude de 99.37%, ce qui signifie qu'il est capable de discriminer presque tous les cas de malwares existant dans l'ensemble de données considéré.

Comme perspective, nous envisageons de raffiner notre étude à travers les points suivants :

- Classification et détection des malwares android par plusieurs classes de famille de malwares.
- combiné d'autres solutions proposées.
- extraire d'autres caractéristiques des applications, et crée un nouveau ensemble de données qui contient plusieurs d'applications.

BIBLIOGRAPHIE

[1] Atelier d'intelligence artificielle. https://github.com/KamiKeys/TalksBlast/blob/master/talentwoman2018/ia.md. consulté le Janury 15,2020.

[2] Intents and intent filters. https://developer.android.com/guide/components/intents-filters. consulté le Janury 22,2020.

[3] Package. https://developer.android.com/reference/java/lang/Package. consulté le Janury 22,2020.

[4] Compréhension des rÉseaux neuraux rÉcurrents (lstm, gru). https://mc.ai/understanding-of-recurrent-neural-networks-lstm-gru/, 2018. consulté le Avril 21,2020.

[5] Keras : la bibliothèque python deep learning. https://keras.io/, 2018. consulté le Mars 21,2020.

[6] What is malware analysis ? defining and outlining the process of malware analysis. https://enterprise.comodo.com/blog/what-is-malware-analysis/, 2018. consulté le Juin 06,2020.

[7] Meilleures applications de pare-feu android pour une meilleure sécurité internet sur les téléphones. https://www.digitalprivatevault.com/blogs/best-android-firewall-apps-for-better-internet-security-on-phones, 2019. consulté le Mars 02,2020.

[8] Numpy. https://numpy.org/, 2019. consulté le Mars 21,2020.

[9] Matplotlib : Visualisation avec python. https://matplotlib.org/, urldate = 2019, 2020. consulté le Mars 21,2020.

[10] pandas python. https://pandas.pydata.org/docs/, 2020. consulté le Mars 21,2020.

[11] Qu'est-ce que scikit-learn ? `https://www.codecademy.com/articles/scikit-learn`, 2020. consulté le Mars 21,2020.

[12] Tensorflow lite guide. `https://www.tensorflow.org/lite/guide`, urldate = 2020-03-31, 2020. consulté le May 20,2020.

[13] Mohammed K. Alzaylaee, Suleiman Y. Yerima, and Sakir Sezer. DL-Droid : Deep learning based android malware detection using real devices. *Computers and Security*, 89, 2020.

[14] Radoniaina Andriatsimandefitra and Ratsisahanana Caract. Caracterisation et detection de malware Android basees sur les flux d'information . Radoniaina Andriatsimandefitra Ratsisahanana. 2015.

[15] Daniel Arp, Michael Spreitzenbarth, Malte Hübner, Hugo Gascon, and Konrad Rieck. Drebin : Effective and Explainable Detection of Android Malware in Your Pocket. (February), 2014.

[16] Saba Arshad, Munam A. Shah, Abdul Wahid, Amjad Mehmood, Houbing Song, and Hongnian Yu. SAMADroid : A Novel 3-Level Hybrid Malware Detection Model for Android Operating System. *IEEE Access*, 6 :4321–4339, 2018.

[17] B. Bathelot. Application android. `https://www.definitions-marketing.com/definition/application-android/`, 2011. consulté le Janury 27,2020.

[18] Cédric Bertrand. *Les malwares sous Android*. 2012.

[19] Elmouatez Billah, Mourad Debbabi, Abdelouahid Derhab, and Djedjiga Mouheb. MalDozer : Automatic framework for android malware detection using deep learning. *Digital Investigation*, 24 :S48–S59, 2018.

[20] Martin Borek, Gideon Creech, and Unsw Canberra. Intrusion Detection System for Android : Linux kernel system calls analysis. 2017.

[21] Y C A Padmanabha Reddy, P Viswanath, and B Eswara Reddy. Semi-supervised learning : a brief review. *International Journal of Engineering & Technology*, 7(1.8) :81, 2018.

[22] D Raúl Lara Cabrera. Machine learning techniques for Android malware detection and classification. (March), 2019.

[23] Ayushi Chahal and Preeti Gulia. Machine learning and deep learning. *International Journal of Innovative Technology and Exploring Engineering*, 8(12) :4910–4914, 2019.

[24] Pedro Domingos. A few useful things to know about machine learning. *Communications of the ACM*, 55(10) :78–87, 2012.

[25] Umer Farooq. Android Operating System Architecture. (July) :2–8, 2018.

[26] Aurélien Géron. *Hands-On Machine Learning with Scikit-Learn and TensorFlow*. O'Reilly Media.

[27] Aurélien Géron. Machine Learning avec Scikit-Learn. page 256, 2017.

[28] Pankaj Gupta, Piyush Kumar, Saksham Wason, and Vishal Yadav. Operating System . 2(2) :37–46, 2014.

[29] Abhilash Hota and Paul Irolla. Deep neural networks for android malware detection. *ICISSP 2019 - Proceedings of the 5th International Conference on Information Systems Security and Privacy*, pages 657–663, 2019.

[30] R Islam and T Mazumder. Mobile application and its global impact. *International Journal of Engineering & ...*, (06) :72–78, 2010.

[31] Haider Khalaf Jabbar and Rafiqul Zaman Khan. Methods to Avoid Over-Fitting and Under-Fitting in Supervised Machine Learning (Comparative Study). (December 2014) :163–172, 2015.

[32] A. G. Khachaturyan and G. A. Shatalov. Elastic Energy of Heterophase Systems of Lamellar Inclusions. *Physics of Metals and Metallography*, 31(6) :1–5, 1971.

[33] Bastien L. Tensorflow : tout savoir sur la bibliothèque machine learning open source. `https://www.lebigdata.fr/tensorflow-definition-tout-savoir`, 2018. consulté le Mars 21,2020.

[34] À L Obtention D E La. Technologies de l ' information m . ing . la gestion de l ' identité fédérée et hiérarchique pour le paradigme.

[35] Bernard Lebel. Analyse de maliciels sur android par l'analyse de la memoire vive. `https://corpus.ulaval.ca/jspui/bitstream/20.500.11794/29851/1/34353.pdf`, 2018. consulté le Janury 27,2020.

[36] Yann Lecun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553) :436–444, 2015.

[37] Solène LIMOUSIN. Android et ios. `https://www.supinfo.com/articles/single/9003-android-ios`, 2019. consulté le Janury 29,2020.

[38] Hongyu Liu and Bo Lang. Machine learning and deep learning methods for intrusion detection systems : A survey. *Applied Sciences (Switzerland)*, 9(20), 2019.

[39] A. Martín, R. Lara-Cabrera, and D. Camacho. A new tool for static and dynamic Android malware analysis. (September) :509–516, 2018.

[40] Alejandro Martín, Raúl Lara-Cabrera, and David Camacho. Android malware detection through hybrid features fusion and ensemble classifiers : The AndroPyTool framework and the OmniDroid dataset. *Information Fusion*, 52 :128–142, 2019.

[41] Abdelmonim Naway and Yuancheng Li. International Journal of Computer Science and Mobile Computing A Review on The Use of Deep Learning in Android Malware Detection. *International Journal of Computer Science and Mobile Computing*, 7(12) :42–58, 2018.

[42] Abdelmonim Naway and Yuancheng Li. International Journal of Computer Science and Mobile Computing A Review on The Use of Deep Learning in Android Malware Detection. *International Journal of Computer Science and Mobile Computing*, 7(12) :42–58, 2018.

[43] Long Nguyen-Vu, Jinung Ahn, and Souhwan Jung. Android Fragmentation in Malware Detection. *Computers and Security*, 87 :101573, 2019.

[44] Chigozie Nwankpa, Winifred Ijomah, Anthony Gachagan, and Stephen Marshall. Activation Functions : Comparison of trends in Practice and Research for Deep Learning. pages 1–20, 2018.

[45] Noeline PAGESY. Application mobile native, web ou hybride ? `https://www.supinfo.com/articles/single/145application-mobile-native-web-hybride`, 2015. consulté le Janury 23,2020.

[46] Dina Saif, S. M. El-Gokhy, and E. Sallam. Deep Belief Networks-based framework for malware detection in Android systems. *Alexandria Engineering Journal*, 57(4) :4049–4057, 2018.

[47] Soumya Sourav, Devashish Khulbe, and Naman Kapoor. Deep Learning Based Android Malware Detection Framework. 2019.

[48] Franklin Tchakounte. A Malware Detection System for Android. *Journal of Chemical Information and Modeling*, 53(9) :1689–1699, 2015.

[49] Christian Urcuqui. Dataset malware/beningn permissions android. `https://www.kaggle.com/xwolf12/datasetandroidpermissions`, 2018. consulté le Mars 20,2020.

[50] Haohan Wang and Bhiksha Raj. On the Origin of Deep Learning. pages 1–72, 2017.

[51] Suleiman Y. Yerima, Sakir Sezer, and Igor Muttik. Android malware detection using parallel machine learning classifiers. *Proceedings - 2014 8th International Conference on Next Generation Mobile Applications, Services and Technologies, NGMAST 2014*, (Ngmast) :37–42, 2014.

[52] Xue Ying. An Overview of Overfitting and its Solutions. *Journal of Physics : Conference Series*, 1168(2), 2019.