REPUBLIQUE DU SENEGAL



Un Peuple – Un But – Une Foi

MINISTERE DE L'ENSEIGNEMENT SUPERIEUR, DE LA RECHERCHE ET DE L'INNOVATION

UNIVERSITE GASTON BERGER DE SAINT-LOUIS



U. F. R DE SCIENCES APPLIQUEES ET DE TECHNOLOGIE

DEPARTEMENT DE PHYSIQUE APPLIQUEE

Mémoire de Fin d'Etudes

En vue de l'obtention du Diplôme de Master d'Ingénierie en Electronique et Télécommunications (MIETEL)

Détection des Malwares basée sur l'Intelligence Artificielle (IA)

Présenté par :

NDEYE SENE

Sous la Direction du :

Pr. CHERIF DIALLO

Année Universitaire 2023/2024

REMERCIEMENTS

Nous tenons à remercier avant tout Dieu,
Le Tout Puissant de m'avoir donné la santé, la patience, et le courage pour réaliser ce modeste travail. Je tiens à exprimer ma profonde gratitude à toutes les personnes qui ont contribué, de près ou de loin, à la réalisation de ce mémoire.

Je remercie tout d'abord mon encadreur Pr Cherif Diallo, pour son accompagnement, ses conseils avisés, sa disponibilité et la confiance qu'il m'a accordée durant tout ce travail. Ses orientations et remarques m'ont permis d'avancer avec rigueur et sérénité.

Je souhaite également remercier l'ensemble des enseignants de l'UFR SAT et responsables de la formation, dont l'engagement et la qualité des enseignements ont nourri et enrichi mon parcours académique.

Mes remerciements s'adressent aussi à mes collègues, camarades de promotion et amis, pour leur soutien moral, leurs échanges constructifs et leurs encouragements constants.

Enfin, j'exprime ma gratitude à ma famille, pour son soutien indéfectible, ses sacrifices, ses prières et son amour qui ont toujours été une source de motivation et de persévérance.

À toutes et à tous, merci du fond du cœur.

DÉDICACES

Je dédie ce mémoire :

À mes parents, particulièrement à ma tendre et chère maman, pour leurs sacrifices, leurs conseils, leurs prières et leur confiance absolue en moi.

À mes sœurs d'amour pour leur présence, leur soutien indéfectible, leur affection et leur motivation de chaque instant.

À ma chère famille, dont le soutien, les encouragements et l'amour inconditionnel m'ont toujours portée.

À mes amis, camarades et promotionnaires, pour leur soutient, leur joie de vivre et de partage

Que ce travail soit le reflet de votre force et de votre amour.

« *Le succès est la somme de petits efforts répétés jour après jour.* »

de Robert Collier

RÉSUMÉ

L'évolution rapide des cybermenaces et la sophistication croissante des logiciels malveillants ont rendu les méthodes traditionnelles de détection, telles que les techniques basées sur les signatures ou l'analyse comportementale classique, insuffisantes face aux malwares modernes polymorphes, métamorphiques et fortement obfusqués. Ce mémoire explore l'apport de l'apprentissage profond dans l'amélioration de la détection automatique des malwares. Après avoir présenté l'état de l'art sur les familles de malwares, leurs techniques d'évasion et les limites des approches classiques ; le travail analyse les modèles modernes de Deep Learning notamment CNN, RNN, LSTM, GRU et MLP en détaillant leurs architectures et principes de fonctionnement. Deux jeux de données représentatifs, EMBER_2018 et CIC-MalMem2022 ont été utilisés pour concevoir, implémenter et évaluer plusieurs modèles, dont un CNN2D, un modèle hybride CNN1D+LSTM, un MLP et un LSTM. Les résultats expérimentaux montrent que les approches basées sur le Deep Learning offrent une performance élevée en termes de précision, rappel et F1-score, avec une capacité accrue à détecter des menaces complexes. Ce travail met en évidence le potentiel du Deep Learning comme alternative robuste aux méthodes classiques et ouvre des perspectives pour l'optimisation des modèles et leur adaptation à des environnements réels.

Mots clés : Malwares, l'apprentissage profonde (Deep Learning), Ember et CIC-Malmem2022

ABSTRACT

The rapid evolution of cyber threats and the increasing sophistication of malicious software have made traditional detection methods; such as signature-based techniques and conventional behavioral analysis, insufficient against modern polymorphic, metamorphic, and heavily obfuscated malware. This thesis investigates the contribution of Deep Learning to improving automated malware detection. After presenting the state of the art on malware families, their evasion techniques, and the limitations of classical approaches, the study examines modern Deep Learning models; including CNN, RNN, LSTM, GRU, and MLP; by detailing their architectures and operational principles. Two representative datasets, **EMBER_2018** and **CIC-MalMem2022**, were used to design, implement, and evaluate several models, including a 2D CNN, a hybrid 1D CNN + LSTM model, an MLP, and an LSTM. Experimental results show that Deep Learning based approaches achieve high performance in terms of accuracy, recall, and F1-score, with an enhanced ability to detect complex threats. This work highlights the potential of Deep Learning as a robust alternative to traditional detection methods and opens promising perspectives for model optimization and adaptation to real-world environments.

Keywords: Malware, Deep Learning, EMBER, CIC-MalMem2022.

TABLE DES MATIÈRES

REMERCIEMENTS I

DEDICACES II

RESUME... III

ABSTRACT IV

LISTE DES FIGURES VIII

LISTE DES TABLEAUX X

SIGLES ET ABREVIATIONS XI

INTRODUCTION GENERALE 1

CHAPITRE1 : ETAT DE L'ART SUR LA DETECTION DES MALWARES 4

1.1 Introduction 5

1.2 Définition des malwares 5

1.3 Typologie des malwares 6

1.3.1 Les virus 7

1.3.2 Vers informatiques (Worms) 7

1.3.3 Cheval de Troie (Trojan Horse) 8

1.3.4 Logiciel espion (Spyware) 8

1.3.5 Les ransomwares 9

1.3.6 Rootkit 9

1.3.7 Keylogger 10

1.3.8 Adware 10

1.4 Les techniques d'évasions des malwares 11

1.4.1 Le polymorphisme 11

1.4.2 Métamorphisme 12

1.4.3 Packers et Crypters 13

1.5 Méthode classique de détection des malwares 14

1.5.1 Analyse statique 15

1.5.1.1 Interprétation abstraite 18

1.5.1.2 Les outils d'analyse statique 18

1.5.2 Analyse dynamique 19

1.5.2.1 Les outils d'analyse dynamique 20

1.5.3 Limites faces aux menaces émergentes 20

1.6. Conclusion 21

CHAPITRE2 : DETECTION DES MALWARES BASEE SUR LE DEEP LEARNING 22

2.1. Introduction 23
2.2. Apprentissage profonde pour la cybersécurité 23
2.3. Définition de l'apprentissage profonde 24
2.3.1. Fonctionnement du DL 24
2.3.2. Classification des méthodes de l'apprentissage profond 25
2.4. Quelques types de modèles de réseaux neuronaux en Deep Learning 26
2.4.1. Deep Neural Network (DNN) 26
2.4.2. Convolutional Neural Network (CNN) 27
2.4.2.1 Convolution Layer 28
2.4.2.2 Pooling Layer 29
2.4.2.3 Fully Connected Layers 30
2.4.3. Recurrent neural networks (RNNs) 30
2.4.3.1 Unités de mémoire à long terme (LSTM) 32
2.4.3.2 Gated Recurrent Unit (GRU) 33
2.4.4. Multi-Layer Perceptron (MLP) 34
2.5. Datasets d'évaluation de détection des malwares basés sur le Deep Learning 35
2.6. Travaux connexes pour la détection des malwares basé sur le DL 36
2.7. Conclusion 43
CHAPITRE3 : CONCEPTION ET REALISATION DES MODELES 44
3.1. Introduction 45
3.2. Environnement d'exécution 45
3.3. Les Datasets utilisés 47
3.3.1. EMBER dataset 47
3.3.2. Le dataset CIC-MalMem2022 49
3.4. Préparation des données 50
3.4.1. Préparation du dataset EMBER 50
3.4.1.1 Filtrage et sélection des données 50
3.4.1.2 Nettoyage et réduction des caractéristiques 51
3.4.1.3 Normalisation et segmentation des données 51
3.4.1.4 Reshape pour les modèles utilisés 52
3.4.2. Préparation du dataset CIC-MalMem_2022 52
3.4.2.1 Nettoyage et sélection des caractéristiques 53
3.4.2.2 Regroupement et classification binaire 53
3.4.2.3 Normalisation et segmentation 53
3.4.2.4 Adaptation des données par rapport au modèle 53
3.5. Architecture des modèles 54

3.5.1. Modèles appliqués au dataset EMBER_2018 55
3.5.1.1 Architecture du modèle CNN2D 55
3.5.1.2 Architecture du modèle hybride CNN1D avec LSTM 57
3.5.2. Modèles appliqués aux jeux de données CIC-MalMem-2022 60
3.5.2.1 Modèle MLP (Multilayer Perceptron) 60
3.5.2.2 Modèle LSTM (Long Short-Term Memory) 61
3.6. Implémentation et résultats des modèles 62
3.6.1. Implémentation des modèles 62
3.6.2. Les résultats obtenus à partir des modèles implémentés 63
3.6.3. Comparaison et analyse globale des performances des modèles 66
3.7. Mesures d'évaluation des modèles 67
3.8. Evaluation sur l'ensemble des données Test 70
3.8.1 Evaluation des modèles à partir des métriques 70
3.8.2 Matrice de confusion 71
3.8.3 La courbe Receiver Operating Characteristic (ROC) 72
3.9. Interprétation générale des résultats 73
3.9.1. Analyse des courbes d'entraînement et de validation : convergence et capacité d'apprentissage 73
3.9.1.1 Résultats sur le dataset EMBER 73
3.9.1.2 Résultats sur le dataset CIC-MalMem2022 74
3.9.2. Interprétation des métriques d'évaluation sur les données test 76
3.9.3. Interprétation des matrices de confusion et des courbes ROC 78
3.9.4. Tableau comparatif final des résultats d'évaluation 79
3.10. Discussion avec les travaux connexes 80
3.11. Conclusion 81
CONCLUSION GENERALE 83
Perspectives : 85

LISTE DES FIGURES

Chapitre1 : Etat de l'art sur la détection des logiciels malveillants

Figure1- 1 Evolution des malwares de 2008 à 2024 [W3] 6
Figure1- 2 : Les types de malwares 6
Figure1- 3 : : Une illustration montrant la variation de positionnement et de niveau d'obscurcissement trouvée dans les malwares métamorphes [6] 13
Figure1- 4 : Taxonomie du système de détection des malwares 15
Figure1- 5 : Le format d'un fichier exécutable [7] 16

Chapitre2 : Détection des malwares basés sur le Deep Learning

Figure2- 1 : Architecture standard d'un modèle Deep Learning [9] 25
Figure2- 2 : Les modèles d'apprentissage du DL [W7] 26
Figure2- 3 : L'architecture d'un modèle CNN [9] 28
Figure2- 4 : Opération de Convolution [9] 29
Figure2- 5 : Pooling Layer [W8] 30
Figure2- 6 : Architecture d'un modèle RNN [W9] 31
Figure2- 7 : Architecture de LSTM [15] 32
Figure2- 8 : Cellule GRU [W11] 34
Figure2- 9 : Architecture du MLP [W12] 35

Chapitre3 : Conception et Réalisation des modèles

Figure 3- 1: Framework les plus populaires du DL [W13] 47
Figure 3- 2: La répartition des données en entrainement et test du dataset ember 48
Figure 3- 3: Échantillons de logiciels malveillants utilisés dans la formation et les tests de classification du dataset CIC-MalMem 49
Figure 3- 4: Les données entraînement et validation après nettoyage 52
Figure 3- 5: l'évolution de la précision (Accuracy) et de la fonction de perte (Loss) pour les ensembles d'entraînement et de validation au fil des époques sur le dataset Ember 65
Figure 3- 6: : L'évolution de l'exactitude et la perte des modèles proposés par rapport aux époques d'apprentissage et de validation sur le dataset CIC-Malmem2022 66
Figure 3- 7: Illustration d'une matrice de confusion[W14] 68

Figure 3- 8: Évaluation comparative des modèles CNN2D, CNN1D+LSTM, MLP et LSTM sur les ensembles de test des jeux de données EMBER et CIC-Malmem2022 pour les métriques : précision, rappel, F1-score et exactitude. 70
Figure 3- 9: Matrices de confusions des modèles CNN2D, CNN1D+LSTM, MLP et LSTM sur les ensembles de test des deux jeux de données (ember et cic-malmem2022). 71
Figure 3- 10: Analyse comparative des performances des modèles de détection de malwares à partir des courbes ROC 72

LISTE DES TABLEAUX

Chapitre1 : Etat de l'art sur les méthodes de détection des malwares

Tableau 1-1: Comparaison de fonctionnement de quelques types de malwares 10
Tableau 1-2: Outils et description utilisés dans l'analyse dynamique 20

Chapitre2 : Détection des malwares basée sur le DL

Tableau2- 1 : Ensembles de données public relatives à la détection des malwares 36
Tableau2- 2 : Tableau comparatif des travaux connexes sur la détection des malwares..... 41

Chapitre3 : Conception et réalisation des modèles de détection

Tableau3- 1: Répartition du dataset ember avant et après filtrage 51
Tableau3- 2 : La représentation structurale du modèle CNN2D..... 56
Tableau3- 3 : La structure du modèle hybride CNN1D+LSTM 57
Tableau3- 4 : La représentation du modèle MLP sur le dataset CIC-Malmem2022 61
Tableau3- 5 : Structure du modèle LSTM 61
Tableau3- 6 : Résumé comparatif des performances des modèles..... 67
Tableau3- 7 : Tableau comparatif final des résultats d'évaluation 80

SIGLES ET ABRÉVIATIONS

AI : **A**rtificial **I**ntelligence

AUC: **A**rea **U**nder the **C**urve

AE: **A**uto-**E**ncoders

API: **A**pplication **P**rogramming **I**nterface

ANN: **A**rtificial **N**eural **N**etwork

CNN: **C**onvolutional **N**eural **N**etwork

DBN: **D**eep **B**elief **N**etworks

DBM: Deep **B**oltzmann **M**achine

DNN: **D**eep **N**eural **N**etwork

DLL: **D**ynamic **L**ink **L**ibrary

DL: **D**eep **L**earning

EMBER: **E**lastic **M**alware **B**enchmark for **E**mpowering **R**esearchers

FP: **F**alse **P**ositive

FN: **F**alse **N**egative

FPR: **F**alse **P**ositive **R**ate

LIEF: **L**ibrary to **I**nstrument **E**xecutable **F**ormats

MLP : **M**ulti-**L**ayer **P**erceptron

MD5: **M**essage **D**igest 5

ML: **M**achine **L**earning

PE: **P**ortable **E**xecutable

RBM: **R**estricted **B**oltzmann **M**achine

ReLU: **R**ectified **L**inear **U**nit

ROC: **R**eceiver **O**perating **C**haracteristic

RNN: **R**ecurrent **N**eural **N**etwork

SVM: **S**upport **V**ector **M**achine

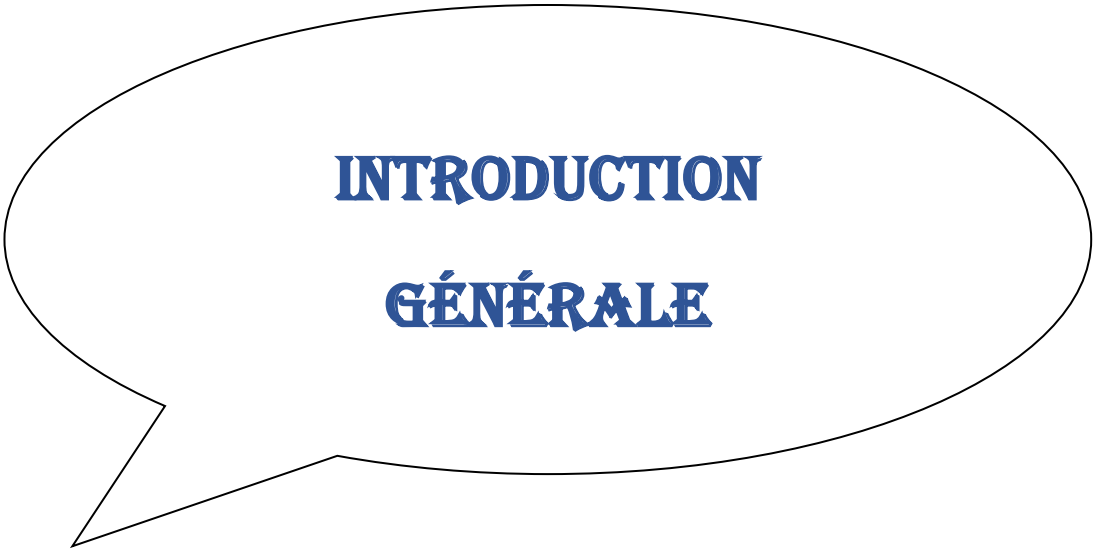
SHA_1: **S**ecure **H**ash **A**lgorithm 1

TP: **T**rue **P**ositive

TN: **T**rue **N**egative

TPR: **T**rue **P**ositive **R**ate

VGG16: **V**isual **G**eometry **G**roup 16


INTRODUCTION
GÉNÉRALE

Introduction générale

L'avènement de l'ère numérique a transformé notre mode de vie avec l'exploration du réseau internet, des technologies connectées et des services de cloud, rendant les technologies de l'information indispensable dans tous les secteurs d'activités. Parallèlement à cette révolution numérique, les cybermenaces se sont multipliées et deviennent plus sophistiquées, mettant en péril la sécurité de nos systèmes d'information avec l'apparition des malwares.

Ces derniers communément appelés logiciels malveillants sont conçus pour infiltrer, dérober ou même endommager les systèmes informatiques en causant des pertes financières considérables, représentent une menace constante et évolutive. La Compagnie AV-TEST[1] déclare avoir collecter plus de 30 000 000 de nouveaux fichiers malveillants en 2012 alors qu'ils étaient à moins de 1 000 000 en 2005. Selon Kaspersky[2] Lab (2016) plus de 6 563 145 hôtes différents ont été attaqué et 4 000 000 objets malveillants ont été détecté en 2015. Récemment, leurs systèmes de détection de menaces ont détecté en moyenne 467 000 fichiers malveillants par jour en 2024, soit une augmentation de 14% par rapport à l'année précédente (Lab, 2024) [W1].

Face à cette menace croissante, les évolutions traditionnelles de sécurité ont atteint leurs limites. Les méthodes de détection basées sur les signatures, sur les comportements ou des heuristiques, bien qu'efficaces pour détecter les malwares connus, se révèlent insuffisantes pour faire face aux nouvelles menaces émergentes comme l'obfuscation, le polymorphisme, le métamorphisme et le packing des malwares. A l'ère de l'intelligence artificielle (IA), cette recrudescence de cyber-malveillance exige désormais une réponse rapide en cybersécurité.

Grâce à ses capacités d'apprentissage automatique et de traitement de quantités de données volumineuses, l'IA offre une approche plus flexible et adaptative pour détecter les logiciels malveillants et ainsi qu'une prévention proactive pour anticiper à des menaces en identifiant les vulnérabilités avant qu'elles ne soient exploitées. Cependant de nombreux enjeux ont été soulevé notamment la qualité des données, le taux de détection qui n'atteint systématiquement pas les 100%, le manque de contexte et la norme légale entre autres.

[1] AV-TEST est l'institut de recherche indépendant pour la sécurité informatique de l'Allemagne.
[2] Kaspersky une entreprise mondiale de cybersécurité et de confidentialité numérique fondé en 1997. Avec plus d'un milliard d'appareils protégés à ce jour contre les cybermenaces émergentes

L'objectif de ce mémoire vise à explorer les différentes techniques de détection de malwares basées sur l'apprentissage profonde, afin d'évaluer leurs performances et d'identifier les défis à relever.

Le présent mémoire est structuré en trois chapitres pour couvrir de manière exhaustive le sujet :

Chapitre1 : Etat de l'art sur les méthodes de détections des malwares : qui explore en détail les malwares, leurs typologies et les méthodes classiques de détection ;

Chapitre2 : Détection des malwares basée sur l'apprentissage profonde : nous mettons en évidence le principe de fonctionnement de l'apprentissage automatique et l'apprentissage profonde, la classification des modèles et d'évaluer quelques datasets et en développant quelques travaux connexes sur ce domaine.

Chapitre3 : La conception et réalisation du modèle : consiste à détailler la méthode d'apprentissage élaborée pour la détection et l'évaluation du modèle.

CHAPITRE1 : ÉTAT DE L'ART SUR LA DÉTECTION DES MALWARES

1.1 Introduction

Les malwares ou logiciels malveillants représentent une menace évolutive pour la sécurité des systèmes informatiques, affectant aussi bien les particuliers que les entreprises et les infrastructures critiques. Ces dernières années, de nombreuses cyberattaques ont fait la une de la presse. A titre d'exemple, le 17 octobre 2024 l'opérateur de télécom **Free** a été victime d'une cyberattaque qui a conduit à l'exfiltration de données personnelles de ses clients dont les coordonnées, numéro téléphone, adresse postale et plus de 5millions de comptes IBAN [W2]. Sous ce même angle, la banque de l'habitat du Sénégal (**BHS**) a subi une attaque le 24 décembre 2024 d'après le magazine **Africa Cybersecurity**, paralysant ainsi plusieurs services dont l'option Bank to Wallet, guichets automatiques plongeant les clients dans le déni total. Ainsi la Direction Générale des Impôts et des Domaines (DGID) du Sénégal est victime d'une cyberattaque le mi-septembre 2025 qui a rendu inaccessibles les principaux systèmes informatiques de l'administration fiscale. Pour contrer ces attaques, plusieurs approches de détection ont été développé au fil du temps allant des méthodes classiques basées sur les signatures et heuristiques aux approches modernes basées sur le Deep Learning.

Dans ce chapitre, nous commencerons par définir les malwares et leur typologie, puis nous explorons en détail les méthodes classiques de détection avant d'aborder leurs limites face aux menaces émergentes.

1.2 Définition des malwares

Le terme « **malware** » une contraction de « **malicious software** » ou logiciel malveillant en français, désigne un ensemble de techniques et d'outils conçus pour infecter et endommager le matériel informatique de leurs victimes, exploiter silencieusement leurs ressources, et extorquer de fond ou exfiltrer des données sensibles et importantes [1]. Kaspersky Labs définit les logiciels malveillants comme « tout type de programme informatique conçu pour infecter l'ordinateur d'un utilisateur légitime et lui infliger des dommages par plusieurs façons » [2]. Historiquement, le premier malware **ElK Cloner** apparût en 1982 a été diffusé via des disquettes aux systèmes **Apple II,** bien que cela n'ait pas nécessairement nui aux ordinateurs, il s'agissait d'une intrusion indésirable, ce qui correspondait au malware que la plupart des gens adoptent. Chaque malware appartient à une famille regroupant ainsi des échantillons sous forme de fichiers exécutables, de scripts ou d'objets injectés en mémoire différents selon

leur mode de fonctionnement et leur objectif. Il existe différents types de malwares dont nous allons explorer en détail dans la partie suivante. L'évolution des malwares par année est représentée dans la figure1-1 suivante selon la compagnie AV-TEST :

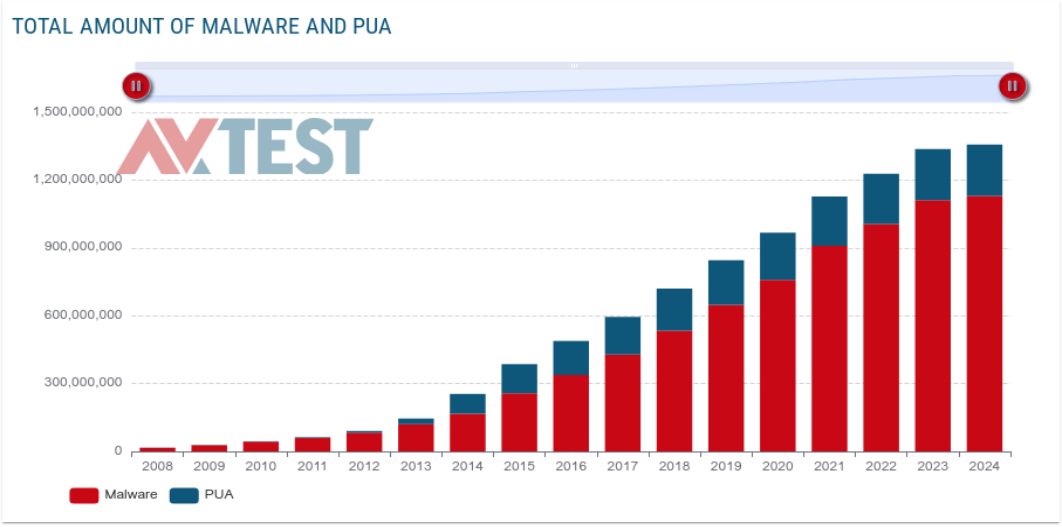


Figure1- 1 Evolution des malwares de 2008 à 2024 [W3]

1.3 Typologie des malwares

Le progrès exponentiel de la technologie ces dernières années a conduit à la multiplication et l'explosion fulgurante des techniques de détection de malwares qui ont permis de détecter un flux massif des types de logiciels malveillants. La figure1-2 ci-dessous représente les types de malwares fréquemment rencontrés.

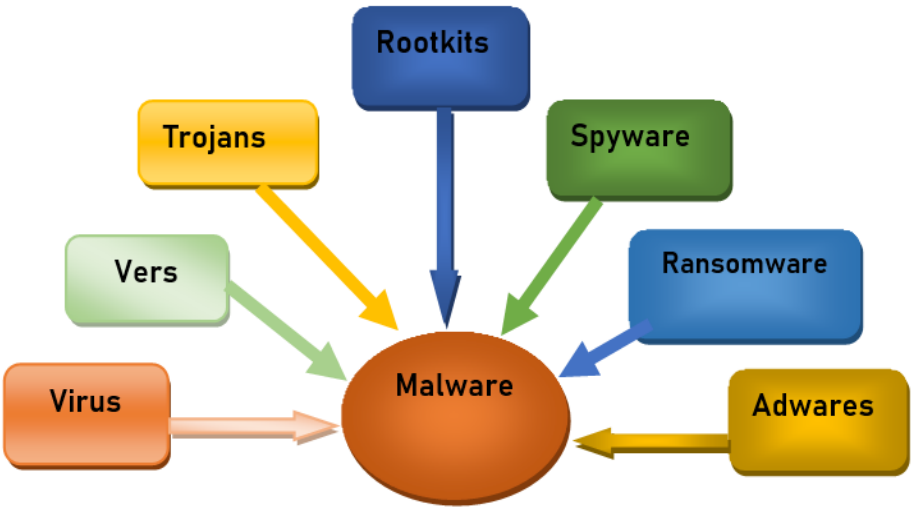


Figure1- 2 : Les types de malwares

1.3.1 Les virus

Les virus font partie des types de malwares les plus connus. Un virus informatique fonctionne en reliant son code malveillant (sa charge utile) à un code propre et attend qu'un utilisateur l'exécute. Ils se répliquent souvent via le partage de fichiers en mode P2P[3], de site internet infectés et aussi le téléchargement de pièces jointes. Une fois qu'un virus pénètre un système, il restera en dommage jusqu'à l'activation du programme ou du fichier hôte infecté, qui a son tour activera le virus et lui permettra de s'exécuter et de se reproduire sur le système. Ils sont souvent cachés dans un fichier exécutable et classés en fonction de la méthode qu'ils utilisent pour se propager dans le réseau [3] :

- **Virus de fichier** : infecte les fichiers exécutables (.exe, .dll, .com, etc) ; lorsqu'un utilisateur ouvre un fichier infecté, le virus s'active et peut se propager à d'autres fichiers du système. Exemple de virus CIH (Tchernobyl[4]).
- **Virus du secteur de démarrage :** infecte le Master Boot Record (MBR) ou la table de partition du disque dur. Il s'active au démarrage de l'ordinateur, avant même le chargement du système d'exploitation. Exemple : **Le virus Michelangelo[5]**
- **Macrovirus :** Cible les fichiers macros comme les documents Word, les feuilles de calcul Excel (.xls), présentations PowerPoint (.ppt). Il utilise les scripts **VBA** (**V**isual **B**asic for **A**pplications) pour se propager. Exemple **Le virus Melissa[6]**
- **Virus de script :** est un malware écrit en langage de script (JavaScript, VBScript, PowerShell, etc.). Il s'exécute souvent via une page web piégée, un e-mail ou un fichier HTML. Exemple : **Le virus ILOVEYOU[7]** [2].

1.3.2 Vers informatiques (Worms)

Les vers informatiques sont des logiciels malveillants qui se propagent sur un réseau informatique comme l'internet. Contrairement aux virus, les vers n'ont pas besoin de programme hôte pour s'exécuter, s'auto-reproduire et se propager. Une fois pénétré, il est capable de se dupliquer et de se propager via le réseau ou la connexion à internet,

[3] P2P : Peer to peer
[4] **Le virus CIH Tchernobyl** : Il s'attaque aux fichiers .exe et peut effacer les données du disque dur.
[5] **Le virus Michelangelo** : Il se déclenche à une date précise et peut rendre le disque dur inutilisable.
[6] **Le virus Melissa** : Il se propageait via des documents Word envoyés par e-mail et infectait des contacts.
[7] **Le virus ILOVEYOU** : Il se propageait par e-mail en pièce jointe et exécutait un script malveillant pour modifier des fichiers.

infectant ainsi tous les serveurs et ordinateurs mal protégés [2] [3]. Ils peuvent exploiter également les erreurs de configuration des réseaux ou exploiter les failles de sécurité des systèmes d'exploitations et applications. Ils sont catégorisés par différentes classes : Ver d'e-mail, Ver de messagerie instantanée, Ver d'IRC[8], Ver de réseau etc.

1.3.3 Cheval de Troie (Trojan Horse)

Cheval de Troie tire son nom de la mythologie grecque, du cheval piégé qui a conduit à la chute de la ville en question, se cachant à l'intérieur d'une structure apparemment innocente, les chevaux de Troie fonctionnent de la même manière, ils semblent être des logiciels légitimes. Ils sont utilisés par des cybercriminels et des pirates informatiques pour accéder aux systèmes informatiques des utilisateurs. Ces derniers sont généralement incités, par le biais d'une technique d'ingénierie sociale (le phishing), à charger et exécuter des chevaux de Troie sur leurs systèmes. Une fois activé, les chevaux de Troie peuvent permettre aux cybercriminels de vous espionner, de dérober vos données sensibles et d'accéder à votre système à l'aide d'un backdoor. Contrairement aux virus, les chevaux de Troie ne se reproduisent pas d'eux-mêmes. À l'inverse, se propage en se faisant passer pour un logiciel ou un contenu utile tout en renfermant secrètement des instructions malveillantes [4].

Emotet[9] est un exemple de cheval de Troie bancaire sophistiqué en circulation depuis 2014. Il échappe aux méthodes de détection basées sur les signatures, est persistant et contient des modules de propagation. Ce cheval de Troie est tellement répandu qu'il fait l'objet d'une alerte émise par le ministère américain de la Sécurité intérieure, qui indique que Emotet a coûté au gouvernement fédéral, ainsi qu'aux administrations locales, et territoriales, jusqu'à 1 million de dollars par incident en mesures correctives.

1.3.4 Logiciel espion (Spyware)

Les logiciels espions sont localisés dans des logiciels gratuits, des pages internet et des sites web. Ils gardent une trace de ce qu'un utilisateur fait sur son ordinateur, se cachent comme un espion à l'arrière-plan de l'ordinateur d'un utilisateur, en recueillant des informations à l'insu de l'utilisateur. Cela peut inclure des données sensibles telles que les détails de la carte de

[8] Ver IRC : est un type de ver qui se répand via les canaux Internet Relay Chats (IRC)
[9] Emotet : détecté pour la première fois en 2014, alors que les clients de banques allemandes et autrichiennes étaient la cible du cheval de Troie

crédit, des mots de passe, des codes PIN, des informations bancaires et des messages non structurés [4].

1.3.5 Les ransomwares

Les ransomwares ou rançongiciels sont des logiciels malveillants qui bloquent l'accès à un ordinateur ou à un réseau entier et empêchent les utilisateurs d'y accéder jusqu'à ce qu'ils paient une rançon. Une fois la rançon payée, l'attaquant fournit des informations sur la manière dont l'utilisateur peut récupérer son système mais y'a aucune garantie de l'envoi de la clé de déchiffrement nécessaire ou que la clé de déchiffrement fournie fonctionnera correctement. Le rançongiciel est créé pour prendre en otage les données personnelles des victimes en leur faisant un chantage [4]. Prenons l'exemple du cas de la ville de Baltimore dont un logiciel rançongiciel nommé « RobinHood[10] » a paralysé toute la ville pendant trois semaines. Près de 10 000 ordinateurs appartenant à la municipalité sont paralysés, entravant les services de la ville et empêchant les habitants de payer certaines de leurs factures en ligne.

1.3.6 Rootkit

Le Rootkit est un type de malware furtif qui permet aux attaquants d'avoir un accès privilégié avec le statut administrateur ou root dans un système tout en restant caché ; ce qui lui permet d'escalader les privilèges, modifier les fichiers d'un système, désactiver les outils de sécurité sans être détecter ou ajouter d'autres malwares [3]. « Zacinlo » est un type de Rootkit, qui infecte les systèmes lorsqu'un utilisateur télécharge une fausse application VPN. Une fois installé, il effectue une analyse de sécurité afin d'identifier et d'éliminer les autres logiciels malveillants. Il ouvre ensuite des navigateurs invisibles et interagit avec les contenus comme le ferait n'importe quel utilisateur, en naviguant, sélectionnant et cliquant. Cette activité a pour but de tromper les logiciels d'analyse comportementale. La charge active de Zacinlo se déclenche lorsque le logiciel malveillant clique sur des publicités dans les navigateurs invisibles. Cette fraude au clic publicitaire permet aux cybercriminels de percevoir une partie de la commission.

[10] RobbinHood un type de logiciel rançongiciel qui crypte les fichiers avec l'algorithme cryptographique RSA-4096. Pour acquérir un outil de décryptage (clé), les victimes de rançongiciels doivent payer une rançon.

1.3.7 Keylogger

Un keylogger a la possibilité d'enregistrer les frappes qu'un utilisateur effectue sur le clavier. Il peut ensuite stocker ces informations et les envoyer à un attaquant. L'assaillant peut ensuite utiliser les données pour déterminer les mots de passe, les noms d'utilisateurs et les détails de paiement. Comme installé par les attaquants, il peut être installé intentionnellement par un administrateur dans une entreprise afin de contrôler les activités des employés [2].

1.3.8 Adware

Adwares ou logiciels publicitaires sont des logiciels qui affichent des programmes indésirables sous forme de fenêtres contextuelles sur un ordinateur ou sur un appareil mobile. Ils peuvent également suivre votre historique de recherche et de navigation afin d'afficher des annonces plus pertinentes. Une fois que le développeur dispose de votre localisation et de l'historique de votre navigateur, il peut gagner de l'argent en vendant ces informations à des tiers.

Comparaison des différents types de malwares avec leurs propriétés :

Le tableau suivant représente un récapitulatif des différents logiciels malveillants et leurs propriétés :

Tableau 1-1: Comparaison de fonctionnement de quelques types de malwares

Malwares / Propriétés	Virus	Vers	Trojan	Ransomware	Rootkit	Spyware	Adware
Autoréplication	✓	✓			✓		
Espionnage	✓	✓	✓	✓		✓	
Composants cachés	✓		✓	✓	✓	✓	
Déni de Service	✓	✓	✓		✓		
Auto-propagation		✓					
Furtivité	✓	✓	✓	✓	✓	✓	
Exécutable	✓	✓	✓	✓	✓	✓	
Insérer une	✓	✓	✓	✓	✓	✓	

charge							
Publicité							✓

Chaque malware a ses propres caractérisques et propriétés ; cependant leur objectif principal reste commun ; infiltrer, voler des données, endommager ou même détruire des systèmes.

1.4 Les techniques d'évasions des malwares

Pour rendre leurs détections difficiles voire impossible par des antivirus, des systèmes de détection d'intrusion (IDS) et des outils d'analyse, les malwares utilisent diverses techniques d'obscurcissement tels que le polymorphisme, le métamorphisme, les packers et Crypters.

1.4.1 Le polymorphisme

Un logiciel malveillant polymorphe est un type de logiciel malveillant qui peut changer de code ou d'apparence à chaque fois qu'il infecte un nouveau système, ce qui rend difficile sa détection par les programmes antivirus traditionnels basés sur les signatures. Cette technique d'évasion consiste à modifier la structure ou la méthode de chiffrement du logiciel malveillant, créant ainsi de nombreuses variantes uniques qui échappent à la détection basée sur les signatures statiques, qui repose sur l'identification de modèles spécifiques dans le code du logiciel malveillant [5]. Selon CrowdStrike, les premiers virus polymorphes sont apparus dans les années 1990 dans le cadre d'un programme de recherche visant à démontrer les limites des antivirus existants. Le tout premier virus de ce type, appelé **polymorphe 1260**[11] ou **V2PX,** qui ajoute du « junk code » à sa routine de déchiffrement, devait servir d'avertissement aux internautes, mais il a aussi malencontreusement inspiré une vague d'activités criminelles basée sur les capacités du virus.

Son principe de fonctionnement est qu'il utilise un compilateur capable de décrypter et d'obscurcir, puis de recompiler le tout. Dans le cas d'un virus, le corps du virus non crypté créait un nouveau décrypteur muté à l'aide d'un algorithme de cryptage aléatoire, puis permettait au décrypteur de se crypter lui-même avant de recompiler les deux sections ensembles [6].

[11] Polymorphe 1260 ou V2PX : est le premier virus polymorphe apparût en 1990 visant à démontrer les limites des antivirus existants, mais malencontreusement il a inspiré à une vague d'activités criminelles basée sur les capacités du virus

Dans ce cas, l'analyse par signature n'est donc plus assez efficace et les antivirus doivent trouver de nouvelles techniques qui permettent de détecter ces malwares. Toutefois, le malware polymorphe a ses limites car son corps après déchiffrement reste le même à chaque itération ce qui donne une possibilité à certains antivirus de le détecter par signature à ce moment-là.

1.4.2 Métamorphisme

Pour se pallier aux manquements des malwares polymorphes, les malwares métamorphes sont capable de se reprogrammer eux-mêmes en modifiant non seulement son apparence mais également son code sous-jacent tout en conservant sa fonctionnalité. Ces modifications peuvent être du renommage de registres, de la permutation de code ou l'ajout de « junk code » par exemple. La sémantique du programme reste ainsi toujours la même tout en possédant un code et donc une signature différente. Le premier malware métamorphe qui utilise ce procédé est **Win95/Regswap**[12] sorti en 1998 [5].

Cette transformation dynamique complique les tentatives de détection, car il n'existe aucune signature fixe à laquelle faire référence, ce qui oblige les systèmes de sécurité à s'appuyer sur une analyse comportementale et des heuristiques plus avancées pour identifier et combattre ces menaces.

La figure1-3 présente un exemple de virus métamorphique. Contrairement au polymorphisme, le code du virus est obscurci, ce qui signifie que l'ensemble du virus est présent dans un état obscurci. Cela pose un problème fondamental puisque « les métamorphes sont des polymorphes de corps » et, par conséquent, n'ont pas de corps constant et renforcent l'idée que la détection basée sur les anomalies est complète.

Cette figure montre comment le malware métamorphe fonctionne pour passer inaperçu à des méthodes de détection. Tout d'abord il se décrypte, ensuite fait son s'obfuscation pour enfin charger son code avant de recompiler le tout. Ce qui fait que sa signature reste inconnue.

[12] Virus métamorphique apparu en décembre 1988 qui utilise la technique de renommage de registres

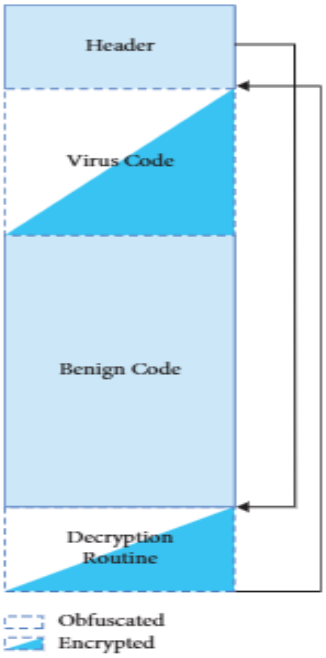


Figure1- 3 : : Une illustration montrant la variation de positionnement et de niveau d'obscurcissement trouvée dans les malwares métamorphes [6]

1.4.3 Packers et Crypters

Les packers et les crypters sont des techniques utilisées par les malwares pour masquer leur code afin d'échapper à la détection basée sur la signature.

Les packers sont des outils qui compressent et cryptent le code du malware, créant un nouvel exécutable qui nécessite l'exécution d'une routine de décompression spécifique, avant de révéler le code malveillant d'origine. Cet obscurcissement rend difficile aux programmes antivirus traditionnels d'identifier la signature du malware, car il est effectivement dissimulé dans l'exécutable compressé [5].

Les crypters, en revanche, se concentrent sur le chiffrement du code du malware et la génération d'une routine de décryptage qui peut reconstituer la charge utile malveillante d'origine au moment de l'exécution. Un crypter contient un stub utilisé pour crypter et décrypter un code malveillant.

Ces deux techniques créent des versions dynamiques et en constante évolution du malware, ce qui rend la détection et le blocage plus difficiles pour les systèmes de sécurité, qui

s'appuient donc davantage sur l'analyse comportementale et l'heuristique pour reconnaître leurs activités malveillantes [5]. La première forme de cryptage a été réalisé par le virus CASCADE[13] en 1980 contenant une routine de chiffrement qui applique un XOR sur son contenu de sorte qu'il apparaisse différemment pour chaque fichier infecté. Le virus a infecté presque tous les ordinateurs d'un bâtiment IBM en Belgique qui a décidé alors de créer son propre antivirus qui est devenu Kaspersky Lab [6].

1.5 Méthode classique de détection des malwares

Au cours de ces dernières années, le nombre de logiciels malveillants rencontrés par les analystes de sécurité a considérablement augmenté. Par conséquent, l'analyse des logiciels malveillants est une discipline essentielle en cybersécurité, consistant à comprendre le fonctionnement, les objectifs et les impacts potentiels des malwares. Cette compréhension approfondie permet de développer des stratégies de défense efficace et de renforcer la résilience des systèmes informatiques. Il existe deux approches fondamentales de l'analyse des logiciels malveillants : l'analyse statique et l'analyse dynamique.

[13] Est un virus informatique historique de type DOS, apparu à la fin des années 1980. Il est célèbre pour son effet visuel spectaculaire : les caractères de l'écran se mettent à "tomber comme une cascade", d'où son nom.

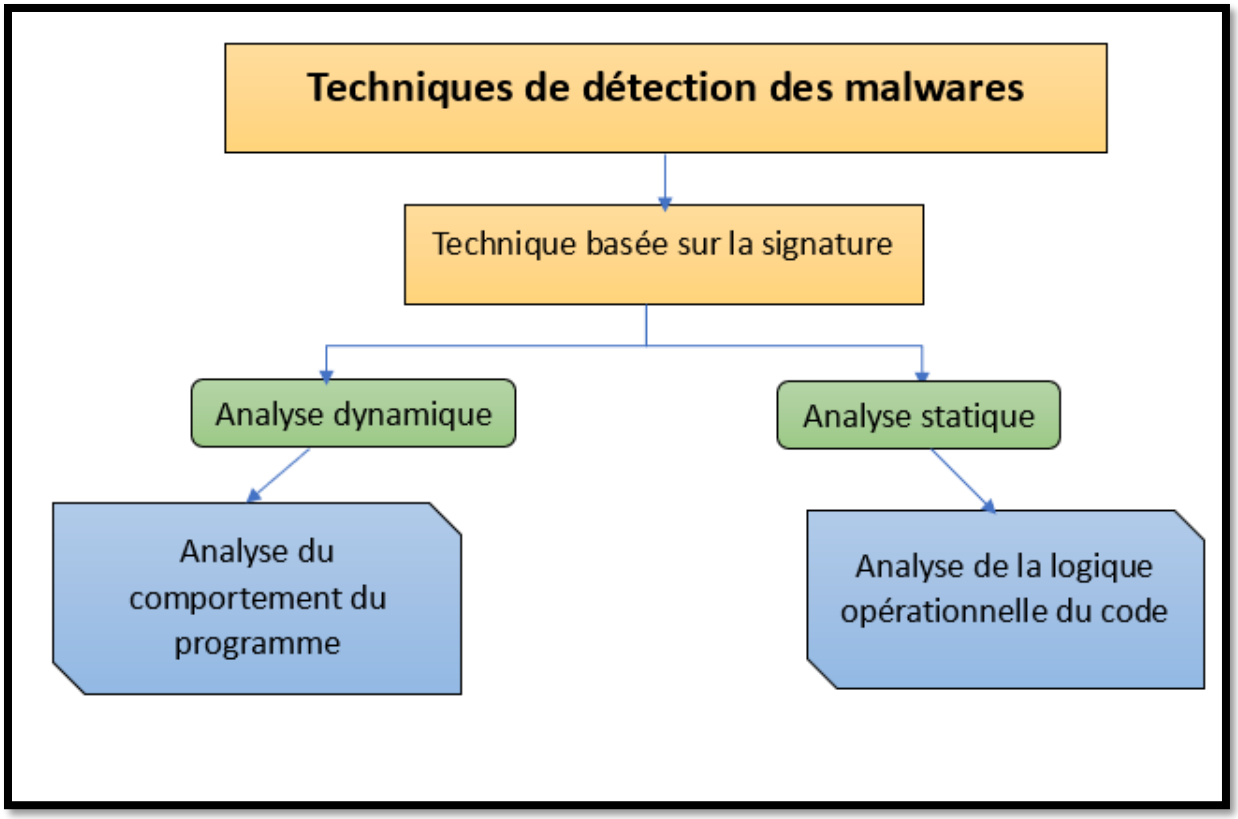


Figure1- 4 : Taxonomie du système de détection des malwares

1.5.1 Analyse statique

L'analyse statique consiste à examiner le code ou la structure du fichier exécutable sans l'exécuter. Ce type d'analyse peut confirmer si un fichier est malveillant, fournir des informations sur ses fonctionnalités et peut également être utilisé pour produire un ensemble simple de signatures. Elle peut révéler des indices sur la nature du malware, tels que les noms de fichiers, les hachages, les adresses IP, les noms de domaines et les données d'en-tête de fichier. La méthode la plus couramment utilisée pour identifier de manière unique un programme malveillant est le hachage. C'est-à-dire qu'un programme de hachage produit un hachage unique, une sorte d'empreinte digitale, qui identifie le programme. Les deux fonctions de hachage les plus populaires sont l'algorithme MD5 et l'algorithme SHA [2].

L'analyse statique est composée de deux grandes familles qui sont la vérification du modèle et l'interprétation abstraite.

La vérification consiste à confirmer qu'un programme respecte ou non un modèle de menace défini, détecter la présence de comportements malveillants dans le code binaire ou source et comparer le programme à une base de signatures existantes pour identifier des malwares connus.

Les approches les plus courantes de l'analyse statique sont notamment :

- **Les fichiers exécutables**

Selon M. Ijaz et al. 47,8% des malwares sont des fichiers exécutables. Dans l'analyse statique, le fichier exécutable est analysé sur la base de sa structure sans l'exécuter dans un environnement contrôlé.

La figure1.5 représente l'entête et la section du fichier exécutable PE :

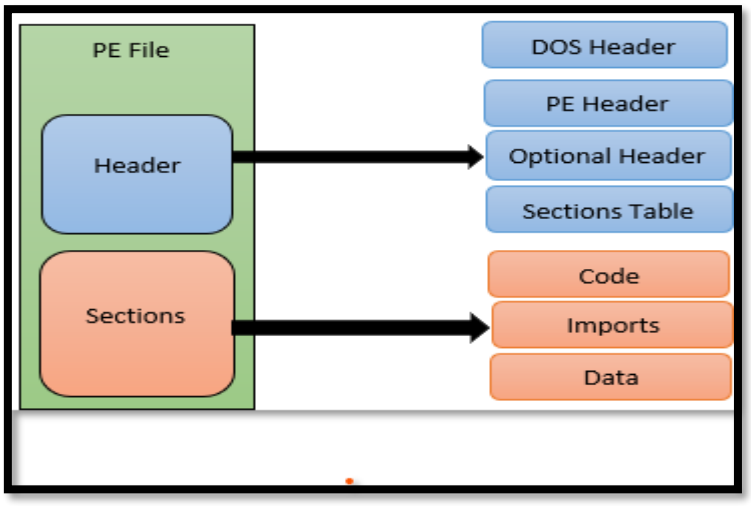


Figure1- 5 : Le format d'un fichier exécutable [7]

Le fichier exécutable possède de nombreux attributs statiques tels que la section et la compacité de la mémoire. PEFILE une bibliothèque python qui extrait les caractéristiques statiques des fichiers exécutables. L'entête est composé de DOS Header dont 17 caractéristiques de base sont extraites de l'en-tête du dossier et deux sont des caractéristiques dérivées, de Optional Header qui au total 40 caractéristiques sont extraites de l'en-tête optionnel et des sections pour l'analyse du fichier PE. Et enfin le PE Header qui donne des informations utiles sur le fichier PE, 9 caractéristiques sont extraites de l'en-tête. La partie « sections » comporte des caractéristiques telles que le code, les importations et les données

utilisée pour déclarer des données initialisées ou des constantes qui ne changent pas au moment de l'exécution [7].

- **Les chaines de caractères présentes dans le binaire**

La recherche dans les chaînes de caractères d'un programme est le moyen le plus simple d'obtenir des indices sur ses fonctionnalités. Les chaînes extraites du fichier binaire peuvent contenir des références aux chemins d'accès aux fichiers modifiés ou accessibles par l'exécutable, aux URL auxquelles le programme accède aux noms de domaine, aux adresses IP, aux commandes d'attaque, aux noms des bibliothèques de liens dynamiques (DLL) Windows chargées, aux clés de registre, etc. L'outil utilitaire « Chaînes de caractères » peut être utilisé pour rechercher des chaînes ASCII (codé sur 8bits) ou Unicode (codé sur 16bits ou plus) en ignorant le contexte et le formatage dans un exécutable. Il utilise la commande « **Strings** » sur Linux/Windows et des outils spécialisés comme **BinText**[14], **FLOSS**[15], ou **PEStudio**[16] [8].

- **Inspection des imports et API Windows**

L'inspection des imports et des appels API Windows sont des étapes essentielles dans l'analyse statique des malwares ; permettent de détecter très rapidement si un fichier exécutable réalise des actions suspectes, comme l'injection de code, la persistance, l'évasion d'antivirus ou la communication avec un serveur malveillant. Les imports sont des fonctions qu'un programme utilise à partir de bibliothèques système comme **kernel32.dll**, **user32.dll**, et **advapi32.dll**.

Les noms de ces fonctions Windows peuvent nous donner une idée de ce que fait l'exécutable. L'utilitaire « *Dependency Walker* » est un programme gratuit pour Microsoft Windows utilisé pour répertorier les fonctions importées et exportées d'un fichier PE [8].

[14] Un outil simple et léger développé par Foundstone, utilisé pour extraire les chaînes de caractères (ASCII, Unicode, ressources, chaînes imprimables) d'un fichier binaire (EXE, DLL, etc.).
[15] Un outil avancé d'extraction de chaînes désobfusquées développé par FireEye. Il est conçu pour extraire non seulement les chaînes visibles, mais aussi celles obfusquées ou générées dynamiquement par les malwares.
[16] Un outil d'analyse statique avancée destiné aux exécutables Windows (PE files).

1.5.1.1 Interprétation abstraite

L'interprétation abstraite est une technique avancée et efficace pour l'analyse statique des malwares. Elle permet de détecter des menaces cachées sans exécuter le programme et est particulièrement utile contre les techniques d'obscurcissement et d'évasion utilisées par les cybercriminels. Son principe fondamental est de remplacer des valeurs concrètes d'un programme à des valeurs abstraites qui englobent tous les cas possibles.

Prenons l'exemple suivant en comparaison avec l'analyse classique :

- Analyse classique :

$$x = 2;$$

$$y = x + 5 \rightarrow y=\mathbf{7}$$

- Analyse par interprétation abstraite

$$x \in [0, 10];$$
$$y = x + 5;$$
Valeur abstraite de y est **[5, 15]**

En résumé, si le programme a des entrées variables, l'analyse abstraite capture tous les cas possibles sans exécuter chaque scénario [2].

1.5.1.2 Les outils d'analyse statique

L'analyse statique est une méthode d'examen des fichiers malveillants sans exécution, en inspectant leur structure, leur code et leurs métadonnées. Différents outils permettent d'effectuer cette analyse en extrayant des informations clés sur le binaire, notamment :

- **PE Studio / Dependency Walker** : pour l'analyse et la détection rapide des fonctions imports et les API Windows suspectes ;
- **IDA Pro / Ghidra** : désassembleurs et décompilateurs permettent une analyse approfondie du code en convertissant le code binaire en code lisible ;
- **Strings / FLOSS** : des outils permettant d'extraire les chaînes de caractères cachées dans un exécutable, révélant des indicateurs de comportement suspect ;
- **DIE / Exeinfo PE** : pour la détection des packers et obfuscation des malwares

- **YARA / VirusTotal** : Identification des malwares connus en comparant son empreinte avec une base de données [W5].

1.5.2 Analyse dynamique

L'analyse dynamique consiste à exécuter le programme et à surveiller son comportement sur le système. Ceci est généralement effectué lorsque l'analyse statique a atteint une impasse, soit en raison de l'obfuscation, soit après avoir épuisé les techniques d'analyse statique disponibles. Contrairement à l'analyse statique, elle retrace les actions réelles exécutées par le programme. Cependant, l'analyse doit être exécutée dans un environnement sécurisé pour ne pas exposer le système à des risques inutiles, où le système est à la fois la machine exécutant, l'outil d'analyse et le reste des machines sur le réseau. À cette fin, des machines physiques ou virtuelles dédiées sont mises en place [8].

Les machines physiques doivent être installées sur des réseaux isolés ; se déconnecter à l'internet ou tout autre réseau pour empêcher la propagation de certains malwares alors que la deuxième option consiste à configurer des machines virtuelles pour effectuer une analyse dynamique. Le système d'exploitation s'exécutant sur la machine virtuelle est isolé du système d'exploitation hôte, de sorte que les programmes malveillants exécutés sur une machine virtuelle ne peuvent pas endommager le système d'exploitation hôte. VMware Workstation, Oracle VM VirtualBox et Microsoft Hyper-V sont quelques-unes des solutions de machines virtuelles disponibles pour les analystes.

Le plus connu des outils d'analyse dynamique est le Cuckoo Sandbox[17], un système d'analyse automatisé des logiciels malveillants open source. Il offre des capacités permettant de tracer les appels d'API, d'analyser le trafic réseau ou d'effectuer une analyse de la mémoire et l'extraction de leur comportement au moment de l'exécution [7].

Les caractéristiques qui peuvent être extraites du rapport Cuckoo Sandbox sont les suivantes : Résumé des informations, Fichiers, Appel d'API pendant l'exécution, Clés de registre, Adresse IP et requêtes DNS et les URL d'accès

[17] Cuckoo Sandbox, un outil développé et maintenu par Claudio Guarnieri dans le cadre du Google Summer of code 2010 et soutenu par le projet Honeynet ; basé sur la virtualisation et l'automatisation des tâches

1.5.2.1 Les outils d'analyse dynamique

Il existe différents outils utilisés pour l'analyse dynamique des logiciels malveillants afin de comprendre leur fonctionnement pour une meilleure prise en compte. Le tableau suivant décrit quelques outils de détection :

Tableau 1-2: Outils et description utilisés dans l'analyse dynamique

Outils	Descriptions
Cuckoo Sandbox	Exécute et analyse les malwares dans un environnement isolé
Any.Run	Sandbox interactive en ligne pour l'analyse des malwares
Process Monitor (ProcMon)	Surveille les processus, le registre et les fichiers
Process Explorer	Visualise les processus actifs et les injections de code
Wireshark	Capture et analyse le trafic réseau du malware
Fiddler	Analyse et manipule les requêtes HTTP/HTTPS
Regshot	Compare l'état du registre avant et après l'exécution du malware
Sysmon (Microsoft)	Capture des logs détaillés sur l'activité du système
OllyDbg / x64dbg	Débogue et analyse l'exécution des binaires malveillants
FakeNet-NG	Simule un environnement réseau pour capturer les requêtes suspectes

1.5.3 Limites faces aux menaces émergentes

L'analyse statique et l'analyse dynamique sont deux approches fondamentales pour l'étude des malwares. Cependant, face aux menaces émergentes, ces méthodes présentent certaines limites qui peuvent compromettre leur efficacité.

L'analyse statique présente des limites avec l'obfuscation des malwares, notamment :

- Inefficace contre les malwares polymorphes et métamorphiques
- Détection contournée si le code est fortement obfusqué avec le chiffrement et la compression
- Ne permet pas d'observer l'exécution réelle des malwares
- Peut renvoyer des faux positifs et des faux négatifs susceptibles de distraire les développeurs.

Cependant l'analyse dynamique malgré ses nombreux avantages par rapport à l'analyse statique, présente aussi des limitations face aux menaces sophistiquées des malwares :

- Les malwares peuvent détecter les environnements de Sandbox et modifier leur comportement
- Peut être dangereux si l'environnement d'analyse est mal configuré pour laisser un exploit de vulnérabilité
- Certaines charges utiles (payload) ne s'exécutent qu'après des conditions spécifiques par exemple activation après plusieurs jours [W6].

1.6. Conclusion

L'évolution constante des malwares représente un défi majeur pour la cybersécurité. Depuis les premiers virus informatiques jusqu'aux menaces sophistiquées actuelles, telles que les ransomwares et les malwares polymorphes et métamorphiques, les attaquants ne cessent de perfectionner leurs techniques d'évasion et d'obfuscation. Cette sophistication croissante a rendu les méthodes traditionnelles de détection de plus en plus limitées. L'analyse statique et l'analyse dynamique bien qu'efficaces pour identifier des signatures connues et des modèles de comportements malveillants, sont vulnérable face aux techniques de chiffrement, de packers, de polymorphisme et de contournement par des techniques anti-sandbox et des malwares multi-étapes.

Face à ces limites, il est devenu impératif d'adopter des approches plus avancées, intégrant l'intelligence artificielle et l'apprentissage profond.

CHAPITRE2 : DÉTECTION DES MALWARES BASÉE SUR LE DEEP LEARNING

2.1. Introduction

L'essor du numérique a conduit à une augmentation exponentielle des cyberattaques, rendant insuffisante les approches traditionnelles de cybersécurité. L'apprentissage automatique un champ d'étude de l'intelligence artificiel (IA), qui vise à donner aux machines la capacité d'apprendre à partir de données via des modèles mathématiques.

Cependant, la machine Learning s'impose comme une solution innovante et efficace. En analysant des ensembles de données massifs et en identifiant des modèles complexes, les algorithmes de ML permettent de détecter et prévenir les menaces en automatisant l'analyse des comportements suspects et la classification des attaques.

Toutefois, les performances de cette méthode reposent implicitement sur la qualité des données d'apprentissage, elles exigent une étape critique appelée « Feature Engineering ». Avec la disponibilité du big-data, un nouveau procès de l'apprentissage automatique appelé le **Deep Learning** (l'apprentissage profond) a été utilisé pour apprendre la représentation et abstraite implicitement les caractéristiques.

Dans ce chapitre, nous allons commencer par l'importance du Deep Learning pour la détection des malwares, ensuite présenté une définition et une classification des diverses méthodes de l'apprentissage profonde. Et à la fin cité différents ensembles de données utilisés pour l'évaluation des techniques de détection des malwares basé sur l'apprentissage profonde.

2.2. Apprentissage profonde pour la cybersécurité

Avec la disponibilité de grandes quantités de données de la cyber infrastructure des réseaux, des systèmes d'exploitation ou des systèmes d'informations et pour relever les défis de la cybersécurité, des méthodes et des techniques comme l'apprentissage automatique (machine Learning), data mining, les statistiques et d'autres capacités interdisciplinaires ont été exploitées [2].

L'apprentissage profonde ou Deep Learning un sous-domaine de l'apprentissage automatique ou ML pourrait être utilisé pour la détection des logiciels malveillants les plus sophistiqués. Ces méthodes de classification et de prédiction peuvent être utilisées pour détecter des motifs et des comportements inhabituels des diverses cyberattaques qui permettent une cyber réponse

en temps réel. Elles ont la capacité de détecter les attaques lorsqu'elle s'est produite et aussi de prédire les futures attaques potentielles [9].

Toutefois, la collecte de données et de trafics réseau ont conduit à un problème de big-data, les experts en sécurité souhaitent toujours de meilleures performances d'un système de détection qui ont un taux de détection le plus élevé et un taux de fausses alarmes le plus bas possible. Par conséquent, les approches de Deep Learning s'adaptent bien à une très grande quantité de données. Ces dernières ont été introduites pour la détection des anomalies de réseau dans le but de différencier les comportements normaux des comportements anormaux afin de détecter des activités malveillantes ou suspectes d'être malveillants [10].

2.3. Définition de l'apprentissage profonde

L'apprentissage profond (*Deep Learning*) est une sous-branche de l'apprentissage automatique (Machine Learning) qui repose sur l'utilisation de réseaux de neurones artificiels composés de nombreuses couches. Ces réseaux, appelés *réseaux de neurones profonds*, sont capables d'apprendre automatiquement des représentations hiérarchiques et complexes à partir de grandes quantités de données [11].

2.3.1. Fonctionnement du DL

L'apprentissage profond repose sur l'utilisation de réseaux de neurones artificiels profonds qui permettent de traiter des données complexes de manière hiérarchique.

Généralement, l'architecture des réseaux profonds est organisée en couches de neurones pour n'importe quel type de ces réseaux : une couche d'entrée (Input Layer), une ou plusieurs couches cachées (Hidden Layers) et une couche de sortie (Output Layer). Chaque paire de couches voisines est connectée, les connexions entre eux appelées poids « weights ». Les neurones d'une même couche généralement appelés « nœuds » n'ont aucune association. La figure 2-1 illustre une architecture standard d'un modèle de réseau de neurones profond [12].

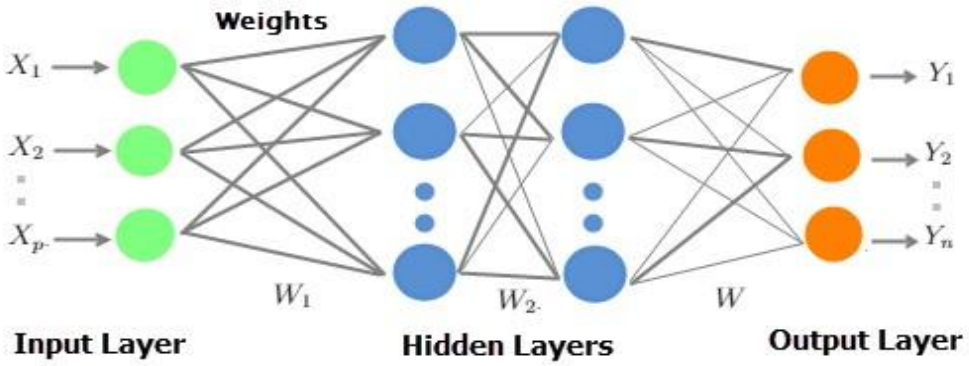


Figure2- 1 : Architecture standard d'un modèle Deep Learning [9]

L'apprentissage profonde se présente comme un système de calcule avancé, c'est l'optimisation des paramètres de poids et le paramètre de biais entre deux couches voisines, Il évalue la justesse du modèle et permet de mieux l'adapter aux données d'apprentissage à ses besoins. Lorsque le modèle arrive à une précision maximale avec des paramètres optimaux, il sera généralisé pour des données réelles. La quantité et la qualité des données d'entraînement déterminent le degré d'apprentissage et donc la précision des modèles obtenus.

2.3.2. Classification des méthodes de l'apprentissage profond

Comme l'apprentissage profond repose sur l'entraînement des réseaux neuronaux à partir de données d'exemple et sur leur récompense en fonction de leur réussite, plus il y a de données, mieux il est possible de construire des structures d'apprentissage profond. Ces approches peuvent être classées en trois modèles, en fonction de la manière dont elles sont formées et destinées à être utiliser :

- **L'apprentissage supervisé** : il est utilisé lorsque les données pour la formation sont clairement étiquetées, il s'agit des réseaux de neurones convolutifs (**CNNs**) et les réseaux de neurones récurrents (**RNNs**) et ainsi que certaines de leurs variantes comme mémoire à long terme (**LSTM**) et unité récurrente fermée (**GRU**).
- **L'apprentissage non-supervisé** : : Il est utilisé lorsque les données d'entrées ne sont pas étiquetées, il s'agit des modèles génératifs visant à regrouper les données selon certains critères de similarité à des fins de reconnaissance ou de synthèse de modèles, à savoir le Deep Belief Networks (**DBN**), Autoencoders (**AE**), Restricted Boltzmann Machine (**RBM**) et Deep Boltzmann Machine (**DBM**).

- **L'apprentissage hybride** : une combinaison de deux méthodes citées ci-dessus. Cette combinaison pourrait fournir une excellente initialisation de base pour laquelle la discrimination pourrait être examiné.

La figure2-2 suivante représente la structure des modèles de l'apprentissage profond :

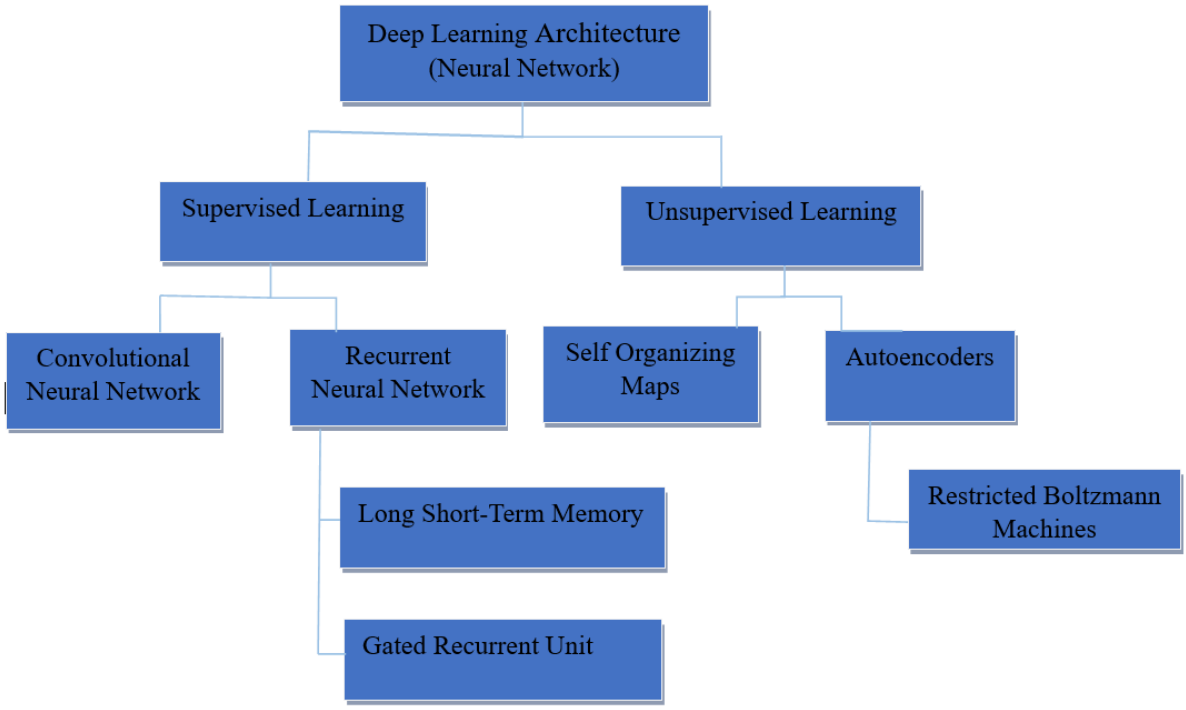


Figure2- 2 : Les modèles d'apprentissage du DL

2.4. Quelques types de modèles de réseaux neuronaux en Deep Learning

L'apprentissage profond repose sur plusieurs architectures de réseaux neuronaux adaptées à différents types de données et de tâches. Ces méthodes permettent d'effectuer des tâches complexes comme la classification, la reconnaissance d'image, la détection d'anomalies et la détection de malwares en cybersécurité. Parmi ces modèles on peut citer :

2.4.1. Deep Neural Network (DNN)

Les Réseaux de Neurones Profonds (DNN) sont un ensemble de neurones organisé en une séquence de couches multiples appelée Multilayer Perceptrons (MLP). Ils se distinguent des réseaux neuronaux traditionnels (Artifical Neural Network-ANN) par leur profondeur et le nombre de couches, de nœuds (neurones) qui composent le réseau.

En général, un réseau neuronal profond comporte une couche d'entrée, une couche de sortie et au moins une couche intermédiaire appelée couche cachée. Plus le nombre de couches est élevé, plus le réseau est profond. Chacune de ces couches effectue différents types de catégorisation spécifiques dans un processus appelé « hiérarchie des caractéristiques ». Le flux du signal d'un tel réseau avec au moins une couche cachée est illustrée par la figure2-1.

Pour la propagation, les données transitent de couche en couche ; traduit mathématiquement par la fonction suivante :

$$y = \delta\left(\sum_{n=1}^{n} w_i x_i + b\right) = \delta(W^T X + b) \quad [9]$$

Avec :

- W : est le vecteur des poids.
- X : est le vecteur des entrées.
- b : désigne le biais.
- δ : représente la fonction d'activation.

2.4.2. Convolutional Neural Network (CNN)

Un CNN est un réseau neuronal multicouche inspiré biologiquement du cortex visuel animal. Son architecture est particulièrement utile dans les applications de traitement d'images, dans l'analyse des features de fichiers PE, des logs système ou des séquences binaires en cybersécurité, des séries temporelles ou des vecteurs de caractéristiques. L'architecture CNN est composée de plusieurs couches qui implémentent l'extraction de caractéristiques puis la classification : des couches de convolution, des couches de groupement (Pooling) et des couches entièrement connectées ; la figure2-3 suivante illustre l'architecture d'un modèle CNN :

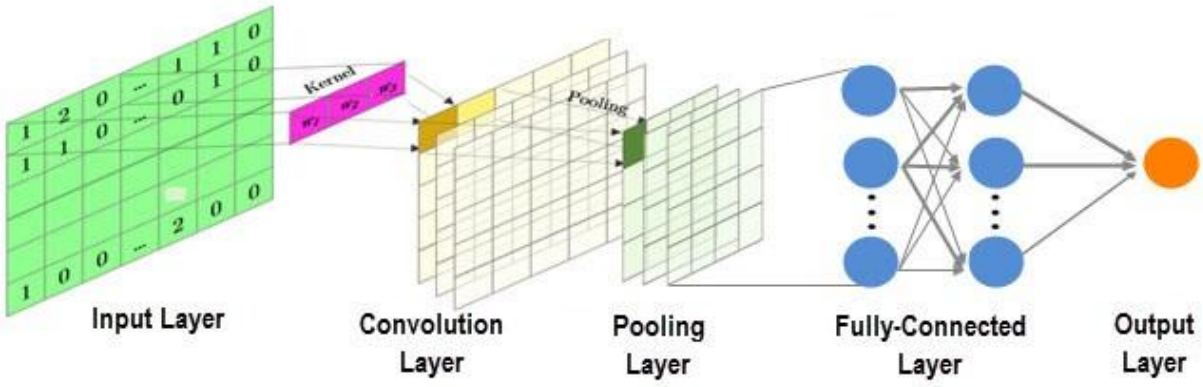


Figure2- 3 : L'architecture d'un modèle CNN [9]

L'image est divisée en champs récepteurs qui alimentent une couche convolutive, qui extrait ensuite les caractéristiques de l'image d'entrée. L'étape suivante est le regroupement (Pooling), qui réduit la dimensionnalité des caractéristiques extraites (par sous-échantillonnage) tout en conservant les informations les plus utiles. Une autre étape de convolution et de regroupement (fully-connected Layer) est ensuite effectuée, qui alimente un perceptron multicouche entièrement connecté. La couche de sortie finale de ce réseau est un ensemble de nœuds qui identifient les caractéristiques de l'image (dans ce cas, un nœud par numéro identifié) [13].

L'utilisation de couches profondes de traitement, de convolutions, de pooling et d'une couche de classification entièrement connectée a ouvert la voie à diverses nouvelles applications des réseaux neuronaux d'apprentissage profond. En plus du traitement d'images, le CNN a été appliqué avec succès à la reconnaissance vidéo et à diverses tâches de traitement du langage naturel.

2.4.2.1 Convolution Layer

Son objectif principal est d'extraire des caractéristiques de l'image d'entrée pour faciliter l'apprentissage du modèle. Elle applique un filtre, un kernel sur une image pour détecter ces caractéristiques spécifiques comme les bords, les textures et les formes. La figure2-4 illustre une opération de convolution à une dimension 1D ; tel que W_i est le filtre appliqué, X_i est l'image d'entrée et C_i la sortie de l'image convoluée [13] :

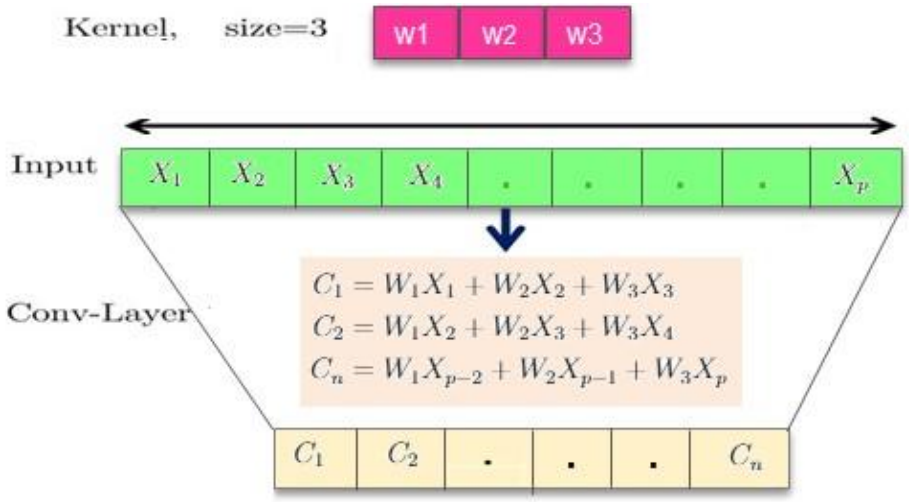


Figure2- 4 : Opération de Convolution [9]

Après la convolution, on applique une fonction d'activation pour introduire de la non-linéarité, essentielle pour apprendre des motifs complexes.

ReLU (Rectified Linear Unit) :

$$f(x)=\max(0,x) \rightarrow \begin{cases} f(x)= x \text{ si } x > 0 \\ f(x)= 0 \text{ si } x \leq 0 \end{cases}$$

Rapide et efficace, remplace les valeurs négatives par zéro. Même si la discontinuité de ReLU à 0 peut nuire aux performances de la rétropropagation, de nombreux travaux ont montré que ReLU fonctionne mieux que les fonctions d'activation « **sigmoïde** » et « **tanh** » de manière empirique [13].

2.4.2.2 Pooling Layer

La mise en commun (Pooling) est un concept important du CNN. Il permet de réduire la charge de calcul en diminuant le nombre de connexions entre les couches convolutives. Dans cette section, nous présentons quelques méthodes récentes de mise en commun utilisées dans les CNN.

- **Max Pooling** (Le plus courant) : prend la valeur maximale d'une région donnée
- **Average Pooling** : utilise la valeur moyenne de chaque groupe de neurones de la couche précédente ;

Le pooling fonctionne en prenant une fenêtre de l'image et en appliquant une opération de réduction sur cette fenêtre. Par exemple, en ne gardant que la valeur maximale (max pooling) ou la moyenne (average pooling) des pixels de la fenêtre. Grâce à cette méthode, le réseau est capable de traiter des images de grande taille tout en restant efficace et précis. Voir figure2-5

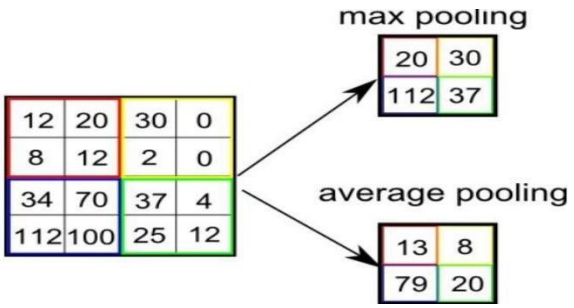


Figure2- 5 : Pooling Layer [W8]

2.4.2.3 Fully Connected Layers

Le voyage des **pixels** dans un réseau de neurones convolutif ne s'arrête pas à la couche de **sortie**. En effet, après avoir traversé les différentes étapes de convolution et de pooling, les données entament une autre partie importante de leur traitement : la transition vers les couches **Fully Connected**.

Ces couches, comme leur nom l'indique, sont totalement connectées. Cela signifie que chaque **neurone** dans une couche Fully Connected est connecté à tous les neurones de la couche précédente. Le but de ces couches est de prendre les caractéristiques de haut niveau, apprises par les couches précédentes, et de les utiliser pour effectuer la classification finale. La sortie de ces couches est généralement traitée par une fonction d'activation. Cette fonction détermine si le neurone doit être activé ou non, en fonction de la somme des produits des poids et des entrées. Cette étape est importante pour la précision de votre réseau neuronal. Elle vous permettra de faire la différence entre une prédiction juste et une prédiction erronée [14].

2.4.3. Recurrent neural networks (RNNs)

Les RNNs sont constitués de plusieurs couches de neurones récurrentes empilées, permettant au modèle d'apprendre des représentations plus abstraites et complexes des données séquentielles. Ils sont conçus pour traiter des données séquentielles (texte, séries temporelles, logs système, etc.). Contrairement aux réseaux de neurones classiques (MLP, CNN), ils

possèdent des boucles internes(mémoire), ce qui leur permet de retenir des informations précédentes et de les utiliser pour prédire la suite d'une séquence. En effet, ce sont des réseaux qui possèdent des **connexions récurrentes** qui leur permettent de **mémoriser et de prendre en compte les informations précédentes** lors du traitement d'une séquence. Ce fonctionnement est mis en place grâce aux boucles dans l'architecture du réseau, comme le montre la figure2-6 [15].

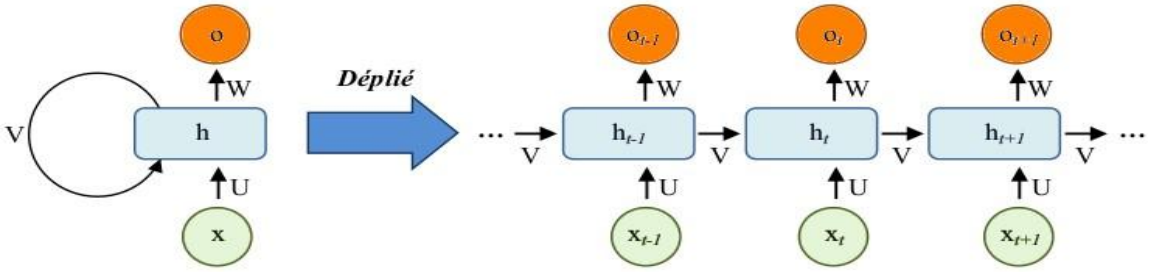


Figure2- 6 : Architecture d'un modèle RNN [W9]

La description mathématique d'un RNN, à chaque instant t, un RNN met à jour son état caché $\mathbf{h_t}$ en fonction de son état précédent h_{t-1} et de l'entrée actuelle x_t :

$$\mathbf{h_t = \delta\,(U\,x_t + V\,h_{t-1} + b_h)}$$

$$\mathbf{O_t = \delta\,(W\,h_t + b_y)\ ;}\ \text{[W9]}$$

La sortie $\mathbf{O_t}$ est obtenue par une transformation de l'état caché

Où :

- h_t : est l'état caché au temps *t*.
- x_t : est l'entrée au temps *t*.
- *U, V, W* : sont les matrices de pondération, Input-to-Hidden, Hidden-to-Hidden et Hidden-to-Output respectivement (connue comme des matrices de transition).
- b_h : est la valeur du biais de l'état caché.
- b_y : est la valeur du biais de sortie.
- O_t : est la valeur de sortie au temps *t*.
- δ : est une fonction de non-linéarité appelée fonction d'activation (soit ReLU ou tanh)

Les RNNs gèrent les séquences et les évènements passés, cependant ils ont des difficultés avec les très longues séquences, les problèmes de gradient. Pour pallier à cela, on utilise ces variantes avec des architectures avancées tels que LSTM et GRU.

2.4.3.1 Unités de mémoire à long terme (LSTM)

Par ailleurs, les cellules LSTM sont plus efficaces pour retenir les informations utiles lors de la rétropropagation du gradient, ce qui permet de corriger les différences entre les prédictions sortantes et les catégories de référence en calculant le gradient de l'erreur pour chaque neurone, en allant de la dernière couche vers la première. La figure2-7 illustre les quatre couches interactives (sigmoïde et tanh), les trois portes (oubli f, entrée i et sortie o) et les opérations pointwise qui traitent le vecteur X à l'intérieur d'une cellule LSTM à un temps t [14].

Les réseaux Long Short-Term Memory sont les plus efficaces pour apprendre des dépendances de long terme et sont donc plus apte à résoudre le problème de la portée discontinue.

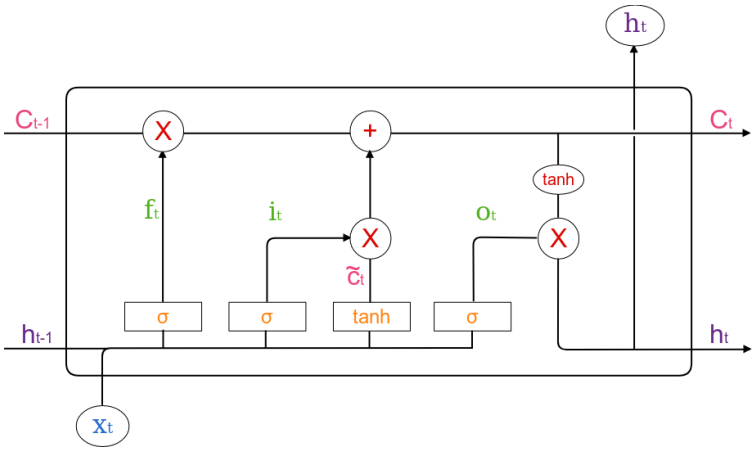


Figure2- 7 : Architecture de LSTM [15]

Le principe de fonctionnement des LSTM est qu'il apprend ce qu'il faut stocker dans l'état cellulaire à long terme, de ce qui peut être oublié par le réseau (état à court terme) et de ce qu'il faut sortir.

Le parcours de l'état à long terme au travers du réseau va de la gauche vers la droite. Il passe d'abord par une porte logique dite d'oubli ($\mathbf{f_t}$) qui laisse de côté certaines informations, puis en rajoute de nouvelles par opération d'addition. Les informations ajoutées sont sélectionnées

par la porte d'entrée ($\mathbf{i_t}$) et le résultat est envoyé à la porte output ($\mathbf{O_t}$). La porte de sortie filtre le résultat via une fonction d'activation tanh qui donne la sortie $\mathbf{h_t}$.

2.4.3.2 Gated Recurrent Unit (GRU)

Une simplification du LSTM a été introduite, appelée Unité Récurrente à Barrière (**GRU**). Ce modèle comporte deux portes, supprimant la porte de sortie présente dans le modèle LSTM. Ces portes sont une porte de mise à jour et une porte de réinitialisation. La porte de réinitialisation (Reset Gate) contrôle la quantité d'informations passées à travers le réseau en décidant les informations précédentes qui doivent être oubliées ou mises à jour. La porte de mise à jour (Update Gate) définit comment incorporer la nouvelle entrée avec le contenu de la cellule précédente c'est-à-dire contrôler quelle mesure l'état caché précédent doit être combiné avec l'activation actuelle afin de former le nouvel état caché. Un GRU peut modéliser un RNN standard simplement en définissant la porte de réinitialisation sur **1** et la porte de mise à jour sur **0** [W10].

La cellule GRU est constituée de deux portes principales, la porte de réinitialisation (**r**) et la porte de mise à jour (**z**), qui contrôlent respectivement l'oubli des informations passées et l'intégration des nouvelles informations dans l'état caché $\mathbf{h_t}$, comme illustré dans la figure2-8. Ce mécanisme permet de capturer efficacement les dépendances temporelles tout en limitant les problèmes de gradient que rencontrent les RNN classiques. Le modèle produit est donc plus simple que le modèle obtenu avec un LSTM standard. En pratique, cette approche permet de réduire le temps de calcul d'un modèle tout en conservant des résultats équivalents à ceux d'un réseau LSTM. Son avantage est sa capacité à mémoriser et à utiliser des informations contextuelles sur des périodes étendues, les rend particulièrement efficaces pour des tâches nécessitant une compréhension approfondie des séquences. Elle offre en outre une capacité de calcul supérieure, qui s'avère idéale pour une application en temps réel ou pour le déploiement sur des appareils ayant des ressources limitées.

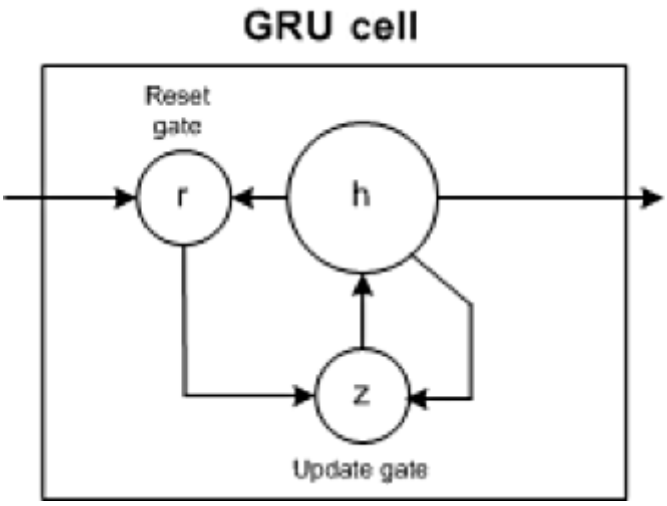


Figure2- 8 : Cellule GRU [W11]

2.4.4. Multi-Layer Perceptron (MLP)

Un Perceptron Multicouche est un type de réseau neuronal feedforward composé de neurones entièrement connectés avec une fonction d'activation de type non linéaire. Il est largement utilisé pour distinguer les données qui ne sont pas linéairement séparables. Sa structure représentée dans la figure2-9 est constituée de couche d'entrée, de couches cachées et de couche de sortie.

Grâce à cette structure, il est capable de modéliser des relations non linéaires complexes et d'effectuer des tâches de classification, régression ou détection, comme dans le cas de la détection de malwares [W12].

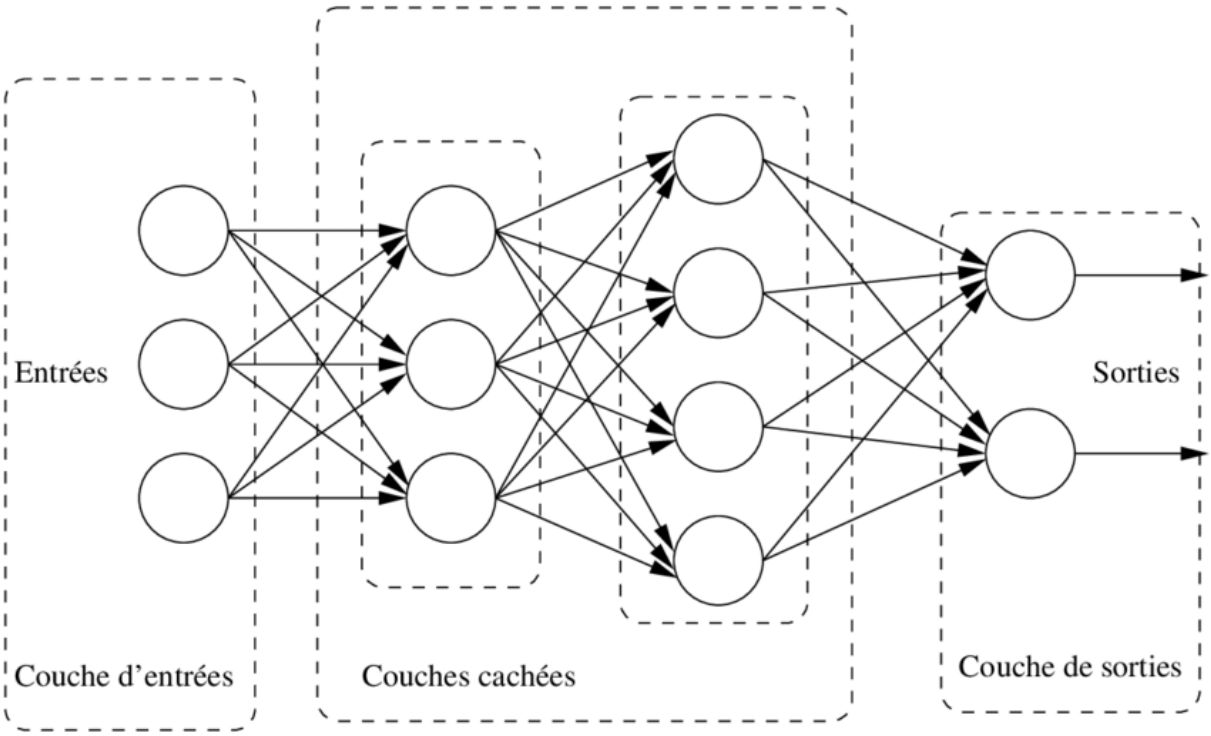


Figure2- 9 : Architecture du MLP [W12]

L'architecture MLP est entièrement connectée, ce qui signifie que chaque neurone d'une couche est lié à tous les neurones de la couche suivante. Cette connectivité complète lui confère un grand pouvoir de représentation, mais au prix d'un nombre parfois très élevé de paramètres.

2.5. Datasets d'évaluation de détection des malwares basés sur le Deep Learning

Plusieurs ensembles de données ont été utilisés dans les travaux de recherche portant sur la détection de malwares par Deep Learning. Ces datasets se distinguent par la nature des données qu'ils contiennent (fichiers PE bruts, features vectorisées, images mémoire, séquences d'appels système ou encore flux réseau) ce qui influence directement le choix des architectures neuronales. Le tableau2-1 suivant résume quelques ensembles de données les plus utilisés ainsi que leurs approches de Deep Learning associées.

Tableau2- 1 : Ensembles de données public relatives à la détection des malwares

Dataset	Type de données	Approche Deep Learning	Année
EMBER	Features PE (vecteurs)	DNN, CNN, RNN	2018
MalMem2022	Images mémoire	CNN, Autoencoders, LSTM	2022
VirusShare	Fichiers PE bruts	CNN, Transformers	2011
Microsoft BIG 2015	Fichiers PE bruts	CNN, RNN	2015
Malimg	Images de malwares	CNN, Vision Transformers	2011
USTC-TFC2016	Captures réseau (PCAP)	CNN, RNN, GRU, LSTM	2016
CDMC 2019	Séquences d'appels système	CNN, RNN, Transformers	2019

2.6. Travaux connexes pour la détection des malwares basé sur le DL

Des travaux antérieurs ont montré que les méthodes DL surpassent d'autres algorithmes d'apprentissage machine tels que la machine à vecteur de support (SVM) et les réseaux artificiels neuronales traditionnels (ANN) dans la détection des anomalies.

L'application de l'apprentissage profond à la détection de malwares a émergé au début des années 2010. En 2011, Nataraj et al. ont proposé une méthode de prétraitement avec laquelle un fichier binaire est converti en image en nuance de gris. Ils utilisent l'algorithme GIST pour extraire les caractéristiques importantes de l'image, cette approche a atteint un taux de succès de 97,25% avec le modèle d'entrainement K-NN dont le but est de classifier les fichiers malveillants en fonction de leur famille [17]. Cette approche a marqué une étape importante dans l'utilisation de l'apprentissage profond pour la détection de logiciels malveillants.

Achi Harrisson et al [1], ont proposé et mis en œuvre un modèle de détection des logiciels malveillants dans les machines. Il s'agit d'un modèle d'apprentissage profond basé sur des réseaux neuronaux artificiels, afin de fournir un modèle très efficace. Ce travail a

permis de développer un modèle de détection de logiciels malveillants en utilisant des données dérivées des caractéristiques des machines. Le langage de programmation Python et la bibliothèque TensorFlow ont été utilisés pour mettre en œuvre ce modèle. Le modèle obtenu, basé sur des réseaux neuronaux, peut prédire la présence ou l'absence de logiciels malveillants avec une précision de 83%. L'utilisation de ce modèle permettra de détecter efficacement les logiciels malveillants, même s'ils peuvent modifier le code pour échapper aux méthodes de détection traditionnelles et éviter les attaques à un stade précoce.

Romain Leveau et al [16], ont exploré l'analyse de malwares et classification à l'aide de réseaux de neurones convolutionnels à une dimension (CNN1D) ; le modèle personnalisé sur l'architecture VGG16[18] présentant une performance légèrement meilleure. En effet, pour le jeu de tests, le résultat issu de ce dernier était de 91,33% de véracité alors que celui obtenu à la suite de l'utilisation du modèle LeNet5 était de 88,68 %. Les résultats se sont donc révélés relativement similaires. Au regard de ces résultats, l'application de réseaux de neurones convolutionnels à un problème de classification entre des maliciels et des fichiers bénins. Quelques petites réserves doivent toutefois être émises : un certain nombre de maliciels ne sont pas correctement classifiés et se retrouvent dans la catégorie des fichiers bénins. Cette constatation peut bien entendu se révéler très problématique en pratique

Benjamin Marais [17], dans sa thèse : « Améliorations des outils de détection de malwares par analyse statique grâce à des mécanismes d'intelligence artificielle », a présenté plusieurs outils relatifs à l'analyse et la détection de fichiers malveillants. Les outils d'analyse préliminaire permettent d'avoir un aperçu de la nature et du comportement d'un fichier. Entre autres, l'extraction d'imports, le taux et le score de malveillances sont de très bons indicateurs de compromission. Pour ce qui est des outils de détection, nous pouvons observer que les modèles entraînés à l'aide de la méthode de prétraitement EMBER sont particulièrement efficaces. Les réseaux de neurones convolutifs produisent des résultats relativement corrects dans l'ensemble, si l'on exclut la classification. La principale faiblesse de cette étude repose sur le manque de données, ce qui est une des pionniers les plus importantes pour l'entraînement de CNN. Notre contribution originale est le modèle bicouches qui permet de répondre à plusieurs problématiques, comme la détection de malware et la détection de

[18] Un modèle de réseau neuronal convolutionnel utilisé pour la reconnaissance d'images

ransomware, à l'aide de sous-modèles optimisés pour ces tâches. De plus, nous pouvons constater qu'il est plus efficace qu'un modèle de classification classique. Le principal avantage de ce modèle est sa flexibilité, qui permet de réentraîner ou d'actualiser un des sous-modèles si l'on dispose de nouvelles données. Il est aussi possible de s'envisager sur d'autres problématiques et d'ajouter d'autres couches à ce modèle selon les besoins. Cette étude s'est principalement concentrée sur la détection de ransomware, mais il serait possible d'adapter la méthode à d'autres catégories de malware comme les fichiers trojans, Worm ou backdoor.

Dans une étude réalisée par Baghirov E.O. et al [18], ils ont présenté une approche basée sur un auto-encodeur pour la détection des logiciels malveillants, en tirant parti de la perte de reconstruction pour identifier les anomalies indiquant une activité malveillante. Les résultats démontrent la capacité du modèle à atteindre un taux de rappel élevé de 91,9 %, ce qui indique son efficacité dans la détection des logiciels malveillants, y compris les menaces nouvelles ou inconnues. L'étude confirme que les auto-encodeurs sont un outil prometteur pour la détection de logiciels malveillants basée sur les anomalies, offrant une alternative évolutive et indépendante de la signature aux méthodes traditionnelles.

Dans une étude réalisée par Halim et al [19], les auteurs ont exploré l'utilisation de réseaux neuronaux récurrents (RNN) pour la détection des logiciels malveillants. En combinant les réseaux à mémoire à long terme (LSTM) et les réseaux neuronaux convolutifs (CNN), le modèle a été conçu pour relever les défis spatiaux et temporels de la classification des logiciels malveillants. Cette approche hybride vise à améliorer la précision de la détection en capturant les schémas complexes du comportement des logiciels malveillants au fil du temps. Cependant, malgré ses résultats prometteurs, le modèle a été confronté à des défis liés à une complexité informatique élevée et à la nécessité de disposer de vastes ensembles de données étiquetées pour une formation efficace.

Panchagnula et al. [20] ont proposé une méthode basée sur l'apprentissage profond pour la détection des logiciels malveillants à l'aide d'auto-encodeurs, où les échantillons de logiciels malveillants sont transformés en images en niveaux de gris pour l'extraction des caractéristiques et la classification. La méthodologie fait appel à deux modèles d'auto-encodeurs : AE-1, qui évalue la faisabilité de la représentation des caractéristiques des logiciels à l'aide d'images en niveaux de gris, et AE-2, qui se concentre sur la classification

des logiciels malveillants par rapport aux logiciels bénins. AE-1 utilise l'apprentissage non supervisé pour l'extraction des caractéristiques, tandis qu'AE-2 intègre l'apprentissage supervisé avec des couches supplémentaires pour la classification. Les performances du modèle ont été évaluées sur plusieurs ensembles de données contenant divers types de logiciels malveillants, avec une précision et des scores F1 élevés. L'approche présente des limites, notamment la dépendance à l'égard des représentations en niveaux de gris qui peuvent négliger les comportements complexes des logiciels malveillants, les coûts de calcul élevés pour l'entraînement des auto-encodeurs et les difficultés potentielles de généralisation à des variantes de logiciels malveillants non vues. Ces facteurs soulignent la nécessité d'apporter d'autres améliorations, telles que l'analyse dynamique ou les méthodes d'ensemble, afin d'améliorer les capacités de détection.

Z. Zhang, dans un article publié en 2022 [21], a utilisé LightGBM, un modèle d'apprentissage automatique. Son travail s'appuyait sur l'ensemble de données Microsoft Malware Prediction, de la base de données Kaggle. LightGBM est un modèle de boosting basé sur l'algorithme de boosting de gradient, conçu pour améliorer les performances et l'efficacité par rapport à d'autres méthodes de boosting telles que XGBoost ou Gradient Boosting Machine (GBM). Il a obtenu un score de 68,4%.

D. Huo et al. [22] ont également mené des recherches sur la détection de logiciels. Ils ont utilisé deux méthodes d'apprentissage, à savoir LightGBM et un CNN unidimensionnel, avec des scores respectifs de 67,16 % et 72,47 %

Dans un autre travail, MeMalDet et al. [23] utilise des auto-encodeurs profonds pour l'extraction de caractéristiques, combinés à un ensemble superposé de classificateurs supervisés pour détecter les logiciels malveillants grâce à l'analyse de la mémoire, en tenant compte des limites des techniques statiques et dynamiques traditionnelles. Bien que la méthode atteigne une précision élevée (98,82 %) et intègre des évaluations temporelles pour des tests réalistes, sa dépendance à l'égard de l'analyse de la mémoire, très gourmande en ressources informatiques, et son manque d'adaptabilité à l'évolution rapide des comportements des logiciels malveillants restent des limites importantes.

Shah, A et al. [24] dans leur article intitulé « Malware Detection Using Deep Learning Approaches ». Cette étude examine comment trois algorithmes d'apprentissage profond les

réseaux neuronaux convolutifs à architecture unidimensionnelle (CNN1D), les réseaux neuronaux profonds (DNN) et un modèle hybride de réseaux neuronaux convolutifs unidimensionnels (CNN1D) et de mémoire à long terme (LSTM) aident à détecter les logiciels malveillants dans les fichiers exécutables portables (.exe). L'efficacité de ces algorithmes à distinguer les exécutables dangereux des exécutables inoffensifs est testée de manière approfondie à l'aide d'un vaste ensemble de données d'échantillons étiquetés de logiciels malveillants et inoffensifs qui est le dataset Ember. En plus de la précision de détection, l'étude évalue également l'efficacité computationnelle de ces modèles sur différents types de matériel (CPU, GPU, Raspberry Pi pour IoT, TPU) afin d'évaluer la faisabilité dans différents environnements. Les scores obtenus sont satisfaisants en fonction du type de ressource.

Orman A. et al. [25] dans leur article sur « Malware Detection Using Memory Analysis Data in Big Data Environment » mettent en avant les limites des méthodes traditionnelles (analyse statique / dynamique) et proposent d'utiliser des données extraites de la mémoire vive dans un environnement big data. Des expériences ont été menées sur l'ensemble de données équilibré CIC-MalMem-2022. La classification binaire a été effectuée à l'aide des algorithmes Random Forest, Decision Tree, Gradient Boosted Tree, Logistic Regression, Naive Bayes, Linear Vector Support Machine, Multilayer Perceptron, Deep Feed Forward Neural Network et Long Short-Term Memory. Les performances des algorithmes utilisés ont été comparées. Les résultats ont été évalués à l'aide des mesures de performance suivantes : précision, score-F1, précision, rappel et AUC. En conséquence, la détection de logiciels malveillants la plus efficace a été obtenue avec l'algorithme de régression logistique, avec un niveau de précision de 99,97 % dans la détection de logiciels malveillants par analyse de la mémoire. Gradient Boosted Tree suit l'algorithme de régression logistique avec une précision de 99,94 %. L'algorithme Naive Bayes a affiché les performances les plus faibles dans l'analyse des logiciels malveillants à partir des données de mémoire, avec une précision de 98,41 %.

Le tableau suivant représente une sélection d'étude portant sur la détection des logiciels malveillants basés sur des modèles d'apprentissage profond, des caractéristiques utilisés, le secteur d'intérêt, les ensembles de données et les mesures de performances.

Tableau2- 2 : Tableau comparatif des travaux connexes sur la détection des malwares

Auteurs	Titre	Modèle	Dataset	Résultat	Année
Orman A. et al.	Malware Detection Using Memory Analysis Data in Big Data Environment	RF, DT, GBT, LR, NB, SVM, MLP, DNN, LSTM	CIC-MalMem-2022 (memory data)	LR ≈ 99.97% acc; GBT ≈ 99.94% acc; NB ≈ 98.41% acc ; LSTM≈99.17% ; MLP≈96.42%	2022
Shah A. & Nawaf L.	Malware Detection Using Deep	CNN1D, NN, CNN1D+LSTM	EMBER	96.4% CNN1D ; ≈96% CNN1D+LSTM; ≈ 97% NN	2024
Panchagnula V. M. et al.	A Deep Learning Approach for Detecting Malware Using Autoencoder	Autoencoder (AE-1, AE-2) + MLP	Plusieurs datasets (Dataset-1, Dataset-2, Dataset-3) — images bytecode	Accuracy 95.8%, TPR ≈ 95.9%	2024
Marais B.	Améliorations des outils de détection par IA (statique)	CNN, bi-couches, attention,	EMBER + datasets internes	CNN ~88%	2023
Nataraj et al.	Malware Classification Using Image-based	GIST + K-NN	Images Binaires	Accuracy 97,25%	2011
Achi Harrison et al.	Malware Detection on Machine Features	ANN/MLP	Données machine	Accuracy 83%	2023
Romain Leveau et al.	Malware Analysis Using 1D CNN (VGG16 vs LeNet5)	CNN1D (VGG16, LeNet5)	Binaires	VGG16 : 91.33%, LeNet5 : 88.68%	2021

Auteurs	Titre	Modèle	Dataset	Résultat	Année
Baghirov E.O. et al.	Malware Detection with Autoencoders	Autoencoder	Données anomalies	Recall 91.9%	2022
Halim et al.	Hybrid RNN/CNN for Malware Detection	LSTM + CNN	Séquences temporelles	Accuracy 98.53%	2019
Z. Zhang	Microsoft Malware Prediction	LightGBM	Microsoft Kaggle	Score 68.4%	2022
D. Huo et al.	Malware Detection	LightGBM, CNN1D	IoT Botnet Dataset	67.16% (LGBM), 72.47% (CNN1D)	2022
MeMalDet et al.	Memory-Based Malware Detection	Deep Autoencoder + Ensemble	MemMal-d2024	98.82 % de précision et 98.72% de Score-F1	2024

2.7. Conclusion

Ce chapitre a présenté le Deep Learning appliqué à la détection des malwares, en détaillant le fonctionnement des réseaux neuronaux et les architectures principales (CNN, RNN, LSTM, GRU, MLP, DNN). Les jeux de données comme EMBER, CIC-MalMem2022 et Microsoft Malware Prediction et autres ont été analysés pour évaluer les performances des modèles. Les travaux connexes montrent que ces approches atteignent des performances élevées, tout en restant limitées par la complexité des modèles, la dépendance aux données et la difficulté à détecter des variantes nouvelles. Ces éléments fournissent une base solide pour la conception et l'implémentation du modèle proposé dans le chapitre suivant.

CHAPITRE3 : CONCEPTION ET RÉALISATION DES MODELES

3.1. Introduction

Dans une logique d'automatisation et de renforcement des capacités de détection des menaces informatiques, ce chapitre est consacré à la conception et à la mise en œuvre de modèle de détection de malwares fondé sur le Deep Learning. L'approche adoptée vise à tirer parti de la puissance des réseaux de neurones pour extraire automatiquement des patterns pertinents à partir de données issues de fichiers exécutables ou de journaux système.

La démarche repose sur une modélisation rigoureuse : après une analyse exploratoire du jeu de données, un pipeline de prétraitement est mis en place afin de normaliser, filtrer et structurer les données en entrée. L'architecture du modèle est ensuite conçue en tenant compte des spécificités des données (dimensionnalité, séquentialité, etc..), avec l'intégration d'un réseau de neurones convolutif (CNN), récurrent (LSTM, ou GRU), ou hybride, selon le cas étudié. L'apprentissage supervisé est utilisé pour tout le long du travail.

Ce chapitre présente en détail l'environnement de développement, les outils de logiciels utilisés comme (TensorFlow, Keras, numpy, etc.), le choix des hyperparamètres, les critères de validation, ainsi que les performances du modèle obtenues sur les données de test. L'ensemble de cette section vise à démontrer la pertinence de l'approche Deep Learning dans le contexte de la détection automatisée des logiciels malveillants.

3.2. Environnement d'exécution

La mise en œuvre d'un modèle de détection de logiciels malveillants basé sur le Deep Learning requiert une puissance de calcul importante. En effet, l'entraînement de réseaux de neurones profonds implique le traitement de grandes quantités de données et l'exécution d'opérations matricielles intensives, qui mobilisent fortement le processeur (**CPU**) et, de manière encore plus marquée, le processeur graphique (**GPU**). Dans ce contexte, le choix de l'environnement d'exécution joue un rôle déterminant dans la réussite et l'efficacité du projet [9]. Deux environnements de travails complémentaires ont été utilisés dans cette partie :

- **Anaconda** : distribution Python largement adoptée en science des données. Elle a permis de configurer facilement des environnements virtuels et d'installer les bibliothèques nécessaires (NumPy, Pandas, Scikit-learn, TensorFlow, Keras, etc.).

Anaconda a été principalement utilisé pour le prétraitement local des données, l'exploration initiale des features et la préparation des jeux d'entraînement et de test.

- **Google Colab** : plateforme cloud gratuite proposée par Google, intégrant un environnement Jupyter Notebook avec un accès aux **GPU et TPU** de 12Go de RAM et plus de 100Go de stockage. Ce choix se justifie par les besoins élevés en ressources de calcul lors de l'entraînement du modèle Deep Learning, qu'un ordinateur classique ne peut généralement pas assurer dans des délais raisonnables. Avec une accessibilité facile, Colab a ainsi permis de bénéficier d'une puissance de calcul adaptée aux expérimentations, réduisant considérablement le temps d'entraînement tout en offrant un environnement flexible et collaboratif.

Le langage de programmation choisit pour le traitement est Python grâce à ces particularités multiplateformes et multiparadigmes et il est aussi appelé le langage le plus populaire pour l'apprentissage automatique et l'intelligence artificielle grâce à sa flexibilité et aussi parce qu'il a un nombre important de bibliothèques logicielles open source disponible. Le développement du modèle de détection de malwares a reposé sur un ensemble de bibliothèques et de Framework Python dédiés au traitement de données, à l'apprentissage automatique et au Deep Learning notamment : Numpy, Pandas, Matplotlib, Seaborn, Scikit-learn, etc. et les frameworks Tensorflow et Keras.

Pandas et Numpy sont utilisés pour la manipulation et le prétraitement des données (la gestion, le nettoyage et la transformation des données tabulaires), Matplotlib et Seaborn pour la visualisation et l'analyse exploratoire et Scikit-learn est utilisé pour le prétraitement et l'évaluation classique qui permet la normalisation des données, la séparation en jeux d'entrainement, de validation et de test, et calcule de métriques de performances. Les Frameworks Tensorflow et Keras utilisés pour la conception et l'entrainement du modèle de Deep Learning. TensorFlow est un Framework open source développée par Google, largement utilisé dans le domaine de l'intelligence artificielle et du Deep Learning. Il est conçu pour le calcul numérique intensif et l'entraînement de modèles de réseaux de neurones, en exploitant efficacement les CPU, GPU et même TPU (Tensor Processing Units). Tandis que Keras intégré dans Tensorflow comme une API de réseau neuronaux écrite en langage de programmation Python. Il s'agit d'une bibliothèque Open Source, exécutée par-dessus des

Frameworks tels que Theano et PyTorch, permettant de concevoir facilement des architectures de réseaux de neurones convolutionels(CNN), récurrents(RNN) ou même hybrides.

La figure3-1 suivante représente les frameworks de Deep Learning les plus populaires avec leurs scores :



Figure 3- 1: Framework les plus populaires du DL [W13]

3.3. Les Datasets utilisés

Dans cette étude, deux jeux de données distincts ont été utilisés afin de diversifier les scénarios d'expérimentation et d'évaluer la robustesse des modèles proposés : **EMBER** et **CIC-MalMem2022.** Le choix de ces datasets se justifie par leur complémentarité : le premier orienté vers l'analyse statique des exécutables, tandis que le second se focalise sur l'analyse dynamique.

3.3.1. EMBER dataset

L'ensemble de données EMBER (Endgame Malware Benchmark for Research), une référence pour les recherches scientifiques, est une collection de caractéristiques à partir de fichier exécutable Windows PE (Portable Executable) incluant à la fois des fichiers malveillants et bénins [W14].

L'ensemble de données Ember2017 contenait 1.1millions de fichiers PE scannés en 2017 ou avant et Ember2018 contient des fonctionnalités de 1millions d'échantillons de fichiers PE en 2018 ou avant, et est répartis en trois catégories : malware (1), bénin (0) et non étiqueté (-1).

Le projet **LIEF** (Library to Instrument Executable Formats) est un outil qui extrait les caractéristiques des fichiers PE dans l'ensemble de données EMBER, les convertit au format **JSON** et les inclut dans l'ensemble de données. Les caractéristiques vectorisées peuvent être créées à partir de caractéristiques brutes et sauvegardées en format binaire, qui peut être converti en CSV ou dans d'autres formats. L'ensemble de données EMBER se compose de huit ensembles de caractéristiques brutes, y compris des caractéristiques analysées, des histogrammes et des décomptes de chaînes. Il se distingue des caractéristiques du modèle, qui sont des caractéristiques vectorisées dérivées de la taille de l'ensemble de données et utilisées pour préparer un modèle. Les caractéristiques brutes sont capturées à l'aide de la technique de hachage unique SHA_256 des caractéristiques. 2381 caractéristiques ont été extraites de l'ensemble de données pour le traitement des caractéristiques. L'histogramme d'octets, l'histogramme d'entropie d'octets, les informations de section, les informations d'importation, les informations d'exportation, les informations générales, les informations d'en-tête, les chaînes d'octets brutes et d'autres aspects pertinents des fichiers d'échantillons de l'ensemble de données sont les paramètres utilisés pour extraire les caractéristiques [24].

Dans cette étude, l'ensemble de données Ember_2018 est utilisé, contenant **1 million** de donnés répartis en données d'entrainement et de test. La répartition des classes des données d'entrainement et de test est représentée dans les figures respectives Figure3-2(a) et Figure3-2(b) ci-dessous.

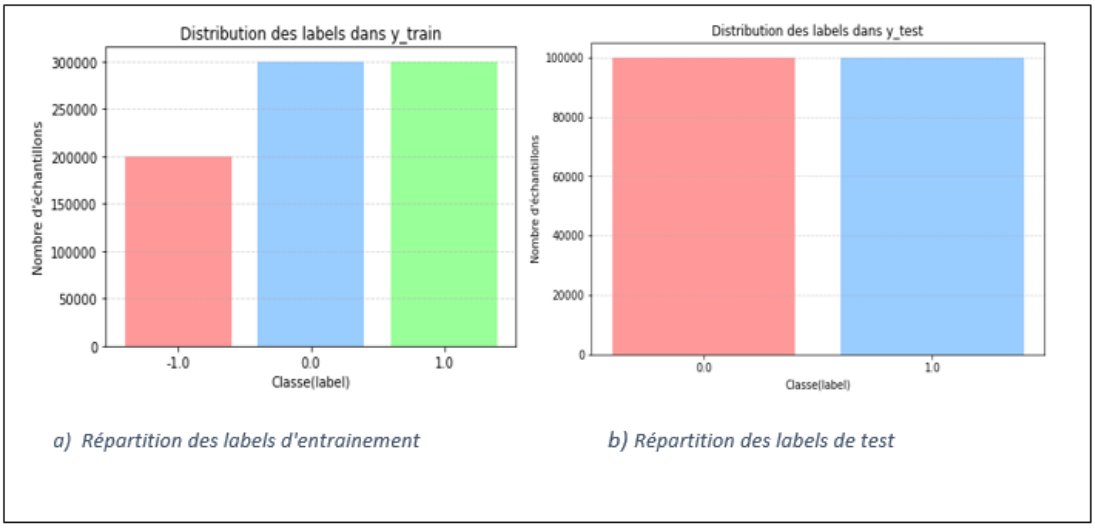


Figure 3- 2: La répartition des données en entrainement et test du dataset ember

La distribution des données d'entrainement et de test de l'ensemble de données ember_2018 est constitué de 800 000 et de 200 000 respectivement en entrainement et test dont répartis 300 000 bénins pour label (0), 300 000 malwares ou label (1) et 200 000 non étiquetés ou label (-1) et pour les données test 100 000 pour label (0) et 100 000 pour label (1).

3.3.2. Le dataset CIC-MalMem2022

L'ensemble de données CIC-MalMem-2022 utilisé dans cette étude a été mis à disposition par l'Institut canadien de cybersécurité en 2022. Cet ensemble de données est conçu pour tester les méthodes de détection des logiciels malveillants obscurcis à l'aide de vidages de mémoire. CIC-MalMem-2022 est un ensemble de données équilibré comprenant au total 58 596 enregistrements dont 57 attributs. Parmi les enregistrements qu'il contient, 29 298 sont bénins et 29 298 sont malveillants. Le vidage de mémoire malveillant a été créé en exécutant des échantillons de logiciels collectés à partir de VirusTotal sur une machine virtuelle dotée d'une mémoire de 2 Go. Ensuite, le comportement normal a été collecté en exécutant des applications sur la machine afin de générer un vidage de mémoire authentique. L'ensemble de données contient trois catégories de logiciels malveillants : les logiciels espions (spyware), les ransomwares et les chevaux de Troie dits Trojans [W15]. La figure3-3 suivante montre la répartition des décharges de mémoire bénignes et malveillantes :

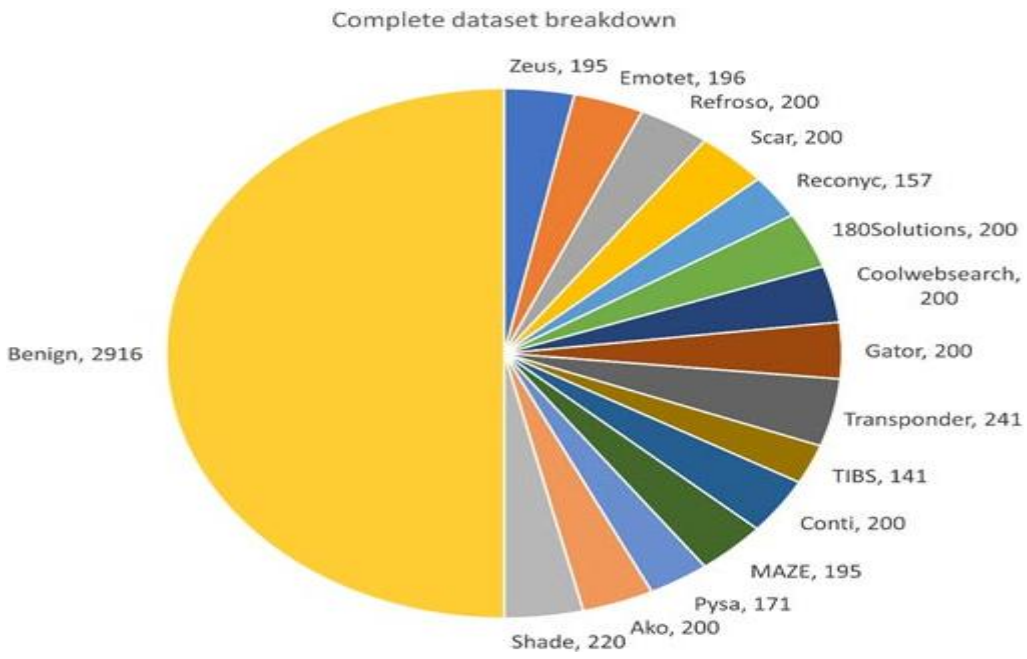


Figure 3- 3: Échantillons de logiciels malveillants utilisés dans la formation et les tests de classification du dataset CIC-MalMem [W15].

3.4. Préparation des données

La phase de préparation des données constitue une étape essentielle avant l'entraînement des modèles de détection de malwares. Elle vise à assurer la qualité, la cohérence et la pertinence des données utilisées pour l'apprentissage automatique. Dans le cadre de ce travail, deux ensembles de données distincts ont été exploités : **EMBER_2018** et **CIC-MalMem-2022**.

Chacun de ces jeux de données a subi des traitements spécifiques, mais une même logique de **classification binaire** (*Benign* / *Malware*) a été adoptée pour assurer l'uniformité de l'analyse et la comparabilité des résultats.

3.4.1. Préparation du dataset EMBER

L'ensemble de données **EMBER_2018** est un dataset de référence pour la détection des malwares à partir des fichiers exécutables Windows PE files. Il contient initialement environ **1 million d'échantillons**, répartis en trois catégories : malware (1), benign(0) *et* unlabeled (-1). Le prétraitement est constitué de plusieurs étapes parmi lesquels :

3.4.1.1 Filtrage et sélection des données

Pour les besoins de ce travail, après la phase de collecte des données qui consiste à extraire et à préparer les détails de l'ensemble de données à partir de fichiers objets JSON. L'ensemble de données a été rigoureusement organisé en différents segments pour l'apprentissage, la validation et les données test afin de permettre une évaluation précise des performances des modèles utilisés. Après le chargement des données, seuls les échantillons labellisés ont été conservés, c'est-à-dire ceux ayant une valeur de `label 0` et de `label 1`. Les lignes dont le label était `-1` ont été supprimées afin d'éviter d'introduire des données non supervisées dans le processus d'apprentissage. Après ce filtrage, les données sont parfaitement équilibrées pour une évaluation fiable des performances du modèle et une réduction du risque de surapprentissage. Le tableau suivant récapitule l'ensemble des données initiales et les données filtrées :

Tableau3- 1: Répartition du dataset ember avant et après filtrage

Dataset Ember	Classe concernée	Nbre d'instances pour l'entrainement avant filtrage	Nbre d'instances pour l'entrainement après filtrage
Classification binaire	MALWARE (1)	300 000	300 000
	BENIGN (0)	300 000	300 000
	Non-étiqueté (-1)	200 000	0

3.4.1.2 Nettoyage et réduction des caractéristiques

Le dataset d'origine comprend **2381** caractéristiques extraites à partir des fichiers exécutables. Après analyse statistique, les colonnes présentant une variance nulle ou constante ont été supprimées, réduisant le nombre final de caractéristiques à **2335**. Cette réduction permet d'éliminer les redondances, de diminuer la complexité du modèle et d'améliorer la rapidité d'entraînement.

3.4.1.3 Normalisation et segmentation des données

Les valeurs numériques des caractéristiques étant sur des échelles différentes, une **normalisation** a été appliquée à l'aide de **StandardScaler**, centrant les données sur zéro et réduisant leur écart-type à 1. Cette étape favorise la convergence rapide des modèles de Deep Learning et empêche que certaines caractéristiques dominent l'apprentissage. Les données ont ensuite été divisé en deux sous-ensembles ; entrainement 80% et 20% de validation. Les résultats après segmentation sont représentés dans l'histogramme suivant :

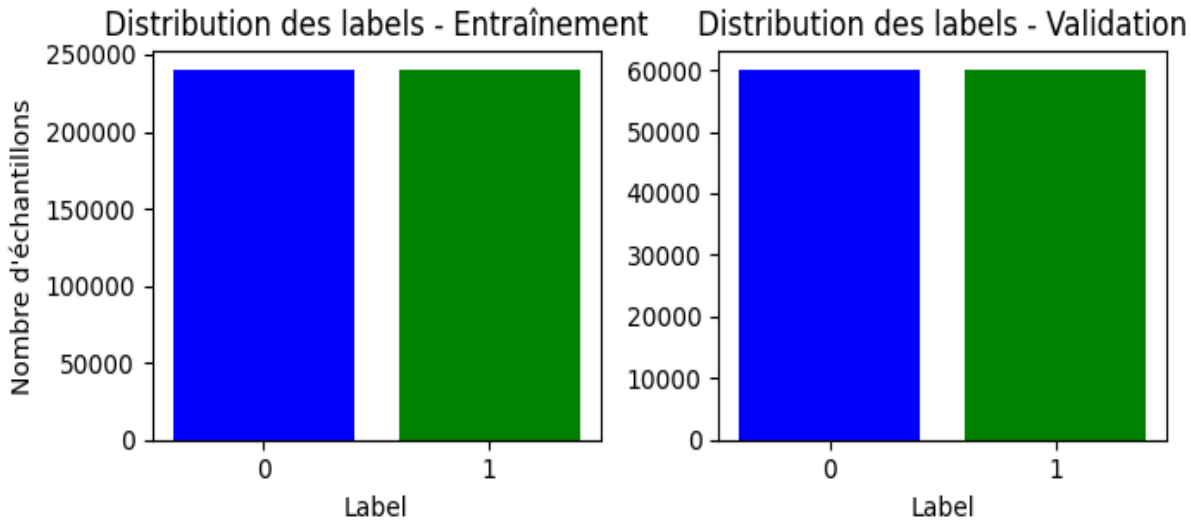


Figure 3- 4: Les données entraînement et validation après nettoyage de ember

3.4.1.4 Reshape pour les modèles utilisés

Selon le type de modèle, la forme des données a été adaptée :

- Pour le **CNN2D**, les vecteurs de 2335 caractéristiques ont été convertis en matrices 2D sous cette forme (47 × 49) afin de permettre l'extraction de motifs spatiaux par convolution bidimensionnelle. Un « padding » a été fait pour compléter la matrice issue des vecteurs 2335 obtenant ainsi ce résultat (49 x49) qui sera utilisé lors de l'entrainement du CNN2D.
- Pour le modèle hybride **CNN1D + LSTM**, les données ont été redimensionnées en séries temporelles unidimensionnelles de forme `(2335, 1)` pour l'extraction séquentielle des patterns à travers les filtres 1D, suivie d'un traitement temporel par les unités LSTM.

3.4.2. Préparation du dataset CIC-MalMem_2022

Pour que l'ensemble de données soit adapté à la classification, certaines étapes de prétraitement sont nécessaires. Ces processus sont importants pour améliorer l'efficacité des modèles de classification, ainsi que pour mettre les données dans un format adapté à l'utilisation d'algorithmes d'apprentissage automatique et d'apprentissage profond.

3.4.2.1 Nettoyage et sélection des caractéristiques

Cette étape de prétraitement consiste à supprimer du jeu de données les caractéristiques qui n'ont aucune incidence sur les performances des algorithmes d'apprentissage. À cette fin, les caractéristiques qui ont une valeur nulle dans l'ensemble de données et n'ont aucun effet sur les résultats des algorithmes d'apprentissage, ont été supprimées de l'ensemble de données. De plus, certaines opérations d'équilibrage des données sont effectuées pour lutter contre le problème de surajustement, en particulier dans les approches d'apprentissage profond. Cependant, l'ensemble de données CIC-MalMem-2022 utilisé dans l'étude est un ensemble équilibré et se compose de deux classes, bénigne et malveillante. Il s'agit d'un ensemble de données résistant au problème de surajustement, car il est équilibré.

3.4.2.2 Regroupement et classification binaire

Dans cette étude, les valeurs de classe catégorielles ont été converties en valeurs numériques à l'aide de Label Encoder. Avec le processus Label Encoder, chaque valeur catégorielle est attribuée de manière aléatoire à une valeur numérique différente, les valeurs catégorielles Bénigne et Malveillante sont attribuées à deux valeurs différentes, 0 et 1, afin de les rendre compatibles avec les algorithmes d'apprentissage automatique et d'apprentissage profond.

3.4.2.3 Normalisation et segmentation

Tout comme le dataset ember, les caractéristiques ont été normalisés afin d'harmoniser les échelles de valeurs et d'améliorer la convergence des modèles. L'ensemble de données a été divisé en deux catégories : 80% pour l'entrainement et 20% pour l'évaluation des données test.

3.4.2.4 Adaptation des données par rapport au modèle

Les données ont été préparées en fonction du modèle d'apprentissage :

- Pour le **MLP (Multi-Layer Perceptron)**, les données ont été conservées sous forme vectorielle (57 colonnes).
- Pour le **LSTM**, les vecteurs ont été transformés en séquences de longueur 57, de forme (57, 1), permettant d'exploiter les dépendances entre les différentes caractéristiques du système.

3.5. Architecture des modèles

L'architecture du modèle joue un rôle déterminant dans la performance du système de détection des logiciels malveillants. Le choix du type de réseau neuronaux dépend principalement de la nature des données que soit statique ou séquentielle et de la complexité des caractéristiques à extraire. Dans ce travail, deux jeux de données distincts ont été utilisés : EMBER_2018, représentant des caractéristiques statiques extraites de fichiers exécutables, et CIC-MalMem-2022, contenant des informations issues de la mémoire des processus.

Les architectures ont été sélectionnées en fonction des propriétés de chaque dataset, afin d'exploiter au mieux leurs structures respectives. Trois types d'approches de réseaux neuronaux CNN, RNN, et le MPL ont été implémenté.

Cependant, ces architectures des modèles proposées partagent certains propriétés et paramètres communs notamment :

- Les couches d'entrées ont les mêmes dimensions que le nombre de caractéristiques sur le vecteur d'entrée pour chaque modèle.
- La fonction d'activation utilisée est « **ReLU » (R**ectified **L**inear **U**nit) pour introduire la non-linéarité et améliorer la propagation du signal dans le réseau, différente à d'autres fonctions comme « **tanh »** et « **sigmoïde »**, car le ReLU a toujours les meilleurs résultats.
- La couche de sortie des modèles implémentés pour la classification binaire est constituée d'un seul neurone activé par une fonction **sigmoïde**. Cette dernière permet de générer une probabilité binaire utilisée pour classifier chaque échantillon comme malware (1) ou bénin (0). Cette fonction est définie par l'expression :

$$\sigma(x) = \frac{1}{1 + e^{-x}} \qquad (1)$$

- Une valeur trop faible n'aura pas d'effet significatif, tandis qu'une valeur trop élevée peut ralentir la convergence ou nuire à la capacité d'apprentissage du modèle qui explique le choix du « **DROPOUT »**. Cette technique a été utilisée pour éviter le problème de sur-apprentissage (Overfitting). Cette technique s'agit de considérer

aléatoirement qu'un pourcentage de neurones d'une couche dans le but d'obtenir un modèle généralisable.

- Pour mesurer l'écart entre les probabilités prédites et les valeurs réelles, la fonction de perte « **Binary Cross-Entropy** » est particulièrement adaptée aux tâches de classification binaire.
- L'optimiseur « **Adam** » a été retenu pour sa capacité à ajuster dynamiquement le taux d'apprentissage, favorisant ainsi une convergence rapide et stable.

3.5.1. Modèles appliqués au dataset EMBER_2018

Le choix du dataset « **EMBER_2018** » s'explique par le fait que ce dernier contient des vecteurs de caractéristiques numériques extraits de fichiers PE, dont la structure est relativement stable et homogène.

Ces caractéristiques peuvent être représentées sous forme de matrices et traitées comme des images numériques, ce qui rend l'utilisation des **réseaux de neurones convolutifs (CNN)** particulièrement adapté.

De plus, pour mieux capturer les corrélations complexes entre les caractéristiques et exploiter les dépendances internes, un **modèle hybride CNN1D** avec **LSTM** a également été proposé.

Ainsi, deux architectures ont été testées : CNN2D et un modèle hybride CNN1D avec LSTM.

3.5.1.1 Architecture du modèle CNN2D

Le modèle **CNN2D** repose sur une architecture classique de réseau neuronal convolutif bidimensionnel. Les vecteurs de caractéristiques du dataset ont été remodelés en matrices afin d'être compatibles avec les couches convolutives.

L'objectif principal du CNN est d'extraire automatiquement des motifs pertinents permettant de distinguer les malwares des fichiers bénins.
La structure du modèle repose sur une succession de couches convolutives, de normalisation et de régulation destinées à extraire et à apprendre les caractéristiques discriminantes entre les fichiers malveillants et bénins.

Le résumé du modèle construit est représenté dans le tableau suivant :

Tableau3- 2 : La représentation structurale du modèle CNN2D

Layer (type)	Output Shape	Paramètres
Conv2d (Conv2D)	(None, 49, 49, 32)	320
Batch_normalization (BatchNo)	(None, 49, 49, 32)	128
Max_pooling2d (MaxPooling2D)	(None, 24, 24, 32)	0
Conv2d_1 (Conv2D)	(None, 24, 24, 64)	18496
Batch_normalization_1 (Batch	(None, 24, 24, 64)	256
Max_pooling2d_1 (MaxPooling2D)	None, 12, 12, 64)	0
conv2d_2 (Conv2D)	(None, 12, 12, 128)	73856
batch_normalization_2 (Batch	(None, 12, 12, 128)	512
max_pooling2d_2 (MaxPooling2	(None, 6, 6, 128)	0
Flatten (Flatten)	(None, 4608)	0
Dense (Dense)	(None, 256)	1179904
Dropout (Dropout)	(None, 256)	0
Dense_1(Dense)	(None, 1)	256
Total params: 1,273,729 Trainable params: 1,273,281 Non-trainable params: 448		

Le modèle implémenté est constitué de plusieurs couches organisées comme suit :

Des couches convolutionelles à deux dimensions Conv2d comportant 32 filtres de taille (3x3) sur les images d'entrées, suivie d'une couche de Batch_normalization qui permet de stabiliser la distribution des activations et d'accélérer la convergence ; complétée par une couche de Max_pooling2D qui réduit la taille des cartes de caractéristiques en conservant les informations les plus importantes de taille (2 x 2). Le deuxième bloc reprend la même structure, mais avec 64filtres renforçant la capacité du réseau à détecter des motifs plus complexes dans les données. Après normalisation et sous-échantillonnage, un troisième bloc de convolution est ajouté avec **128 filtres**, permettant au réseau d'extraire des caractéristiques

de haut niveau et d'augmenter la profondeur d'apprentissage. Chaque couche de convolution utilise la fonction d'activation **ReLU,** qui introduit la non-linéarité nécessaire à la modélisation de relations complexes tout en évitant les problèmes de gradient nul. Une fois ces opérations terminées, les cartes de caractéristiques produites par la dernière couche de pooling sont aplaties à l'aide d'une couche **Flatten**, transformant les tenseurs multidimensionnels en un vecteur unidimensionnel de 4608 éléments. Ce vecteur sert d'entrée à une couche **Dense** de 256 neurones, chargée de combiner les informations extraites pour la classification finale. Pour prévenir le surapprentissage, une couche de **Dropout** est intégrée à ce niveau, désactivant aléatoirement une partie des neurones à chaque itération d'entraînement. Enfin, la **couche de sortie** est constituée d'un seul neurone activé par la **fonction sigmoïde**, qui convertit la sortie en une probabilité comprise entre 0 et 1. Cette valeur traduit la fiabilité du modèle dans le fait qu'un échantillon donné soit malveillant. Le modèle compte au total **1 273 729 paramètres**, dont la majorité sont entraînables.

3.5.1.2 Architecture du modèle hybride CNN1D avec LSTM

Le modèle hybride **CNN1D** avec **LSTM** a été conçu afin de tirer parti des avantages combinés des **réseaux convolutionels** et des **réseaux de mémoire à long terme**. En effet, la convolution 1D permet d'extraire efficacement les caractéristiques locales pertinentes des séquences de données issues du dataset **EMBER**, tandis que la couche **Bidirectional LSTM** (BiLSTM) capture les dépendances temporelles entre ces caractéristiques dans les deux directions (passé et futur) renforçant ainsi la capacité du modèle à comprendre les relations contextuelles dans les séquences de caractéristiques. La structure du modèle choisie est représentée dans le tableau suivant :

Tableau3- 3 : La structure du modèle hybride CNN1D+LSTM

Layer (type)	Output Shape	Paramètres
input_1 (InputLayer)	[(None, 500, 1)]	**0**
conv1d_13 (Conv1D)	(None, 15, 128)	4224
batch_normalization_19 (Batc	None, 15, 128)	512
dropout_19 (Dropout)	None, 15, 128)	0
conv1d_14 (Conv1D	(None, 15, 128)	49280

Layer (type)	Output Shape	Paramètres
batch_normalization_20 (Batc	(None, 15, 128)	512
max_pooling1d_12 (MaxPooling	(None, 7, 128)	0
dropout_20 (Dropout)	(None, 7, 128)	0
bidirectional (Bidirectional	(None, 128)	98816
batch_normalization_21 (Batc	(None, 128)	512
max_pooling1d_12 (MaxPooling	(None, 7, 128)	0
dropout_20 (Dropout	(None, 7, 128)	0
bidirectional (Bidirectional	(None, 128)	98816
batch_normalization_21 (Batch	(None, 128)	512
dense_14 (Dense)	(None, 256)	33024
dropout_21 (Dropout)	(None, 256)	0
dense_15 (Dense)	(None, 128)	32896
dropout_22 (Dropout)	(None, 128)	0
dense_16 (Dense)	(None, 1)	129
Total params: 219,905 Trainable params: 219,137 Non-trainable params: 768		

On voit à l'entrée du modèle un tenseur de dimension **500** qui est obtenu en retirant les colonnes constantes et en faisant la sélection des features les plus importantes avec RandomForest ce qui permettent d'avoir un entrainement plus rapide et une évaluation plus fluide. Une taille de (500, 1) a été choisie après une phase de mise en forme et de normalisation des données afin d'assurer une compatibilité optimale avec les couches convolutionelles et récurrentes.

Le réseau débute par une première couche convolutionelle unidimensionnelle **(Conv1D)** comportant **128 filtres**, avec une taille de noyau égale à **64** et un pas (**stride**) de **32**. Cette couche extrait les motifs locaux présents dans les vecteurs de caractéristiques, tout en appliquant la fonction d'activation **ReLU** pour introduire la non-linéarité nécessaire à la

modélisation de relations complexes. La sortie de cette couche est ensuite normalisée à l'aide d'une **couche Batch Normalization**, qui standardise les activations sur chaque mini-lot, stabilisant ainsi le processus d'entraînement. Une **couche Dropout** est également intégrée à ce stade, désactivant aléatoirement une fraction des neurones afin de réduire le risque de surapprentissage.

Une deuxième **couche Conv1D** est ensuite ajoutée, également composée de **128 filtres** mais avec une taille de noyau plus réduite (**3**) et un pas de **1**, permettant d'affiner les motifs extraits lors de la première convolution. Cette couche est à nouveau suivie d'une **normalisation par lots (Batch Normalization)**, puis d'une **couche de sous-échantillonnage MaxPooling1D**, qui réduit la dimension spatiale des cartes de caractéristiques en conservant uniquement les informations les plus représentatives. Une autre couche **Dropout** vient renforcer la régularisation avant le passage à la partie séquentielle du modèle.

La sortie des couches convolutionelles est ensuite transmise à une **couche bidirectionnelle LSTM (Bidirectional LSTM)** composée de **64 unités** dans chaque direction, soit un total de 128 unités. Cette couche joue un rôle essentiel en apprenant les dépendances temporelles dans les deux sens de la séquence, améliorant la capacité du modèle à détecter des schémas récurrents ou des anomalies caractéristiques d'un comportement malveillant. Une **Batch Normalization** est également appliquée à la sortie de cette couche afin de normaliser les valeurs et d'accélérer la convergence de l'apprentissage.

La partie finale du modèle est constituée de **trois couches entièrement connectées (Dense)**. La première, composée de **256 neurones** avec la fonction d'activation **ReLU**, fusionne les caractéristiques extraites et les projette dans un espace de dimension plus compacte. Elle est suivie d'une **couche Dropout** pour limiter le surapprentissage. La deuxième couche dense contient **128 neurones**, également activée par ReLU, permettant un raffinement supplémentaire des représentations internes. Une autre couche Dropout vient compléter cette étape. Enfin, la **couche de sortie** est une couche **Dense** à **un seul neurone**, activée par la fonction **Sigmoïde**, qui produit une probabilité indiquant si l'échantillon analysé est **bénin (0)** ou **malveillant (1)**.

L'ensemble du modèle contient **environ 220 000 paramètres entraînables**, ce qui représente un compromis adéquat entre capacité d'apprentissage et efficacité computationnelle. L'architecture hybride ainsi conçue combine la capacité de la convolution à détecter des motifs discriminants et celle du LSTM à interpréter des relations temporelles complexes, rendant le modèle particulièrement performant pour la détection de logiciels malveillants à partir de caractéristiques extraites.

3.5.2. Modèles appliqués aux jeux de données CIC-MalMem-2022

Ce dataset, riche en dimensions et en attributs statistiques, permet d'entraîner des modèles d'apprentissage profond capables de distinguer les traces malveillantes des traces légitimes. Dans le cadre de cette étude, deux modèles d'apprentissage profond ont été mis en œuvre : un **réseau de neurones multicouche (MLP)**, adapté aux données tabulaires statiques, et un **réseau LSTM (Long Short-Term Memory)**, plus approprié pour la capture de dépendances temporelles dans les séquences mémoires.

Les architectures détaillées de ces modèles sont présentées ci-dessous.

3.5.2.1 Modèle MLP (Multilayer Perceptron)

Le premier modèle implémenté est un réseau de neurones multicouche (**MLP**), qui représente une architecture de base en apprentissage profond. Ce type de réseau est constitué de couches entièrement connectées (Dense), où chaque neurone d'une couche est relié à tous ceux de la couche précédente.

L'objectif du MLP est d'apprendre des **représentations hiérarchiques** à partir des caractéristiques extraites du dataset, sans tenir compte des relations temporelles entre les échantillons. Afin d'améliorer la performance et de réduire le surapprentissage, des couches de **normalisation (BatchNormalization)** et de **régularisation (Dropout)** ont été intégrées entre les couches denses.

La structure finale du modèle MLP est résumée dans le tableau3-4 ci-dessous :

Tableau3- 4 : La représentation du modèle MLP sur le dataset CIC-Malmem-2022

Layer Input	Output Shape	Paramètres
dense_10 (Dense)	(None, 64)	3 584
batch_normalization_2 (BatchNormalization)	(None, 64)	256
dropout_6 (Dropout)	(None, 64)	0
dense_11 (Dense)	(None, 32)	2 080
batch_normalization_3	(None, 32)	128
dropout_7 (Dropout)	(None, 32)	0
dense_12 (Dense)	(None, 1)	33
Total params: 6,081 (23.75 KB) Trainable params: 5,889 (23.00 KB) Non-trainable params: 192 (768.00 B)		

3.5.2.2 Modèle LSTM (Long Short-Term Memory)

Le second modèle mis en œuvre repose sur un réseau **LSTM,** une variante du réseau neuronal récurrent RNN. Ce type d'architecture est particulièrement adapté à la modélisation des **séries temporelles** et des **séquences de données corrélées dans le temps**, comme celles issues de l'observation du comportement dynamique d'un processus en mémoire.

Le modèle LSTM utilisé est composé de deux couches récurrentes empilées, suivies de couches de normalisation, de régularisation et de couches denses finales. Sa structure est illustrée dans le tableau3-5 suivant :

Tableau3- 5 : Structure du modèle LSTM

Layer (type)	Output Shape	Paramètres
lstm_3 (LSTM)	(None, 55, 128)	66 560
batch_normalization_3(BatchNormalization)	(None, 55, 128)	512
dropout_5 (Dropout)	(None, 55,128)	0
lstm_4 (LSTM)	(None, 64)	49 408
batch_normalization_4	(None, 64)	256
dropout_6 (Dropout)	(None, 64)	0
dense_4 (Dense)	(None, 64)	4 160
batch_normalization_5	(None, 64)	256

Layer (type)	Output Shape	Paramètres
dropout_7 (Dropout)	(None, 64)	0
dense_5 (Dense)	(None, 1)	65
Total params: 121,217 (473.50 KB) Trainable params**:** 120,705 (471.50 KB) Non-trainable params **:** 512 (2.00 KB)		

Dans ce modèle, la première couche LSTM capte les **caractéristiques dynamiques** à court terme des séquences mémoires. La seconde couche LSTM condense ces informations en un vecteur global décrivant les dépendances à long terme. Les couches BatchNormalization améliorent la stabilité de l'entraînement, tandis que les Dropout évitent le surapprentissage du modèle. Enfin, la couche dense finale effectue la classification binaire des échantillons.

3.6. Implémentation et résultats des modèles

3.6.1. Implémentation des modèles

Dans cette étude, quatre modèles CNN2D, CNN1D+LSTM, MLP et LSTM ont été mis en œuvre, entrainés et évalués sur les jeux de données respectifs EMBER_2018 et CIC-MalMem2022. Plusieurs tests ont été fait afin d'obtenir de bonnes Hyperparamètres pour chaque modèle. Ces Hyperparamètres ne peuvent pas être ajustés durant la phase de l'apprentissage et pourtant, ils ont un grand impact sur les performances de modèle durant l'apprentissage. Ils comprennent les variables qui déterminent la structure du réseau (Nombre de neurones, Nombre de couches, fonction d'activation), le lot d'échantillons (Batch Size) et le nombre d'itérations etc… Chaque modèle a été configuré pour une classification binaire (malware / bénin) avec la fonction de perte binaire logarithmique (*binary cross-entropy*) et l'optimiseur **Adam**, reconnu pour sa rapidité de convergence expliqués lors des architectures des modèles.

Les expériences ont été exécutées sur **Jupyter notebook** et **Google Colab**, qui offrent un environnement GPU optimisé, essentiels à l'entraînement des réseaux de neurones profonds. La mise en œuvre des modèles est répartie en deux parties constituées comme suite :

- Dans la première expérimentation qui a été faite sur le sous-ensemble de données Ember_2018 indiquée dans la figure3-4. Les données d'entraînement ont été divisées

en 2 parties : 80% pour l'apprentissage et 20% pour la validation. L'apprentissage ne prend pas beaucoup de temps, le modèle CNN2D a été formé sur environ 10 itérations tant disque le modèle hybride est entrainé sur 50 époques. Les résultats obtenus ont une exactitude très bonne **98%** pour le CNN2D et **97%** pour le modèle hybride CNN1D avec du LSTM. Nous notons ici que les 2 modèles convergent vers une valeur de perte minimale en apprentissage et en validation. Ce qui indique que ces modèles seront généralisés bien au-delà de l'ensemble d'apprentissage. Ensuite, ces modèles seront évalués sur l'ensemble des données de tests.

- La deuxième et dernière expérimentation qui a été faite en utilisant le jeu de données CIC-MalMem2022 indiqué sur la figure3-3. Les données de formation ont été segmentées en 2 parties : 80% pour l'entrainement (apprentissage et validation) et 20% pour l'évaluation du test. L'apprentissage ne prend pas beaucoup de temps, les modèles MLP et LSTM ont été formés dès 15 à 20 époques. Pour les modèles, nous notons que l'exactitude d'apprentissage et de validation augmente constamment du début à la fin, elle atteint une valeur maximale qui tend vers **1**. Nous notons aussi que la valeur de perte diminue fortement durant l'entraînement et l'évaluation et atteint une valeur minimale tendant vers **0**. Cela signifie que ces modèles apprennent mieux et effectuent des meilleures prédictions après chaque époque d'optimisation.

3.6.2. Les résultats obtenus à partir des modèles implémentés

Les courbes d'évolution de la **précision (accuracy)** et de la **fonction de perte (loss)** ont été tracées pour visualiser le comportement des modèles durant l'entraînement. Les résultats de l'évolution des performances au cours de l'entrainement sont représentés dans les figure3-5 et figure3-6.

La figure3-5 représente l'évolution des courbes de l'Accuracy et de la Loss pour le modèle CNN2D et le modèle hybride combinant le CNN1D avec du BiLSTM sur le dataset EMBER et la figure3-6 est la représentation des performances des modèles MLP et LSTM utilisant le jeu de données CIC-Malmem-2022.

- Dans la figure3-5(a), on observe une augmentation régulière de la précision aussi bien sur l'ensemble d'entraînement que sur l'ensemble de validation, atteignant respectivement **98% et 97%** après dix époques. La courbe de perte décroît de manière significative, traduisant une convergence rapide du modèle. L'écart relativement faible

entre les courbes d'entraînement et de validation montre une bonne généralisation, ce qui confirme la capacité du CNN2D à extraire efficacement les caractéristiques pertinentes des fichiers binaires. La combinaison de couches **BatchNormalization** et **Dropout** a contribué à limiter le surapprentissage tout en stabilisant le processus d'optimisation.

- Dans la figure3-5(b), On constate une convergence rapide dès les premières époques, avec une précision d'environ **97 %** sur l'ensemble de validation et une perte inférieure à **0,15** dès la 20^{e} époque.

L'intégration d'une couche **BiLSTM** après les convolutions à une dimension a permis de capter les dépendances séquentielles entre les caractéristiques extraites, améliorant la stabilité et la performance du modèle. La proximité entre les courbes d'entraînement et de validation traduit une excellente capacité de généralisation et une faible variance, signe d'un apprentissage équilibré.

Ces résultats confirment que la combinaison convolutionnelle et récurrente offre une modélisation plus robuste que le CNN seul, en exploitant à la fois les motifs spatiaux et les relations temporelles des séquences dérivées des exécutables.

- Les résultats d'entraînement du réseau de neurones multicouche (MLP) présentés dans la **figure 3.6(c)**. Dès les premières époques, le modèle affiche une hausse rapide de la précision, atteignant presque **100 %** sur les deux ensembles (entrainement et validation) avant la dixième époque. La **fonction de perte** diminue de manière considérable, passant de **6%** à moins de **0,5%,** traduisant une **très bonne convergence** du modèle. L'écart quasi nul entre les courbes d'entraînement et de validation montre que le modèle n'est pas affecté par le overfitting, ce qui signifie qu'il généralise efficacement aux nouvelles données.

Cette performance remarquable s'explique par la simplicité du jeu de données et la capacité du MLP à apprendre les **relations non linéaires** entre les variables après une normalisation correcte. Les couches **BatchNormalization** et **Dropout** ont également contribué à stabiliser l'apprentissage et à limiter la variance du modèle.

- La **figure 3.6(d)** illustre les courbes d'évolution du modèle **LSTM** entraîné sur le même jeu de données CIC-Malmem2022. Le modèle montre également une **convergence très rapide**, avec une précision approchant les **99,9 %** dès les premières époques et une perte qui chute en dessous de **0,01** après cinq époques. Le faible écart entre les courbes d'entraînement et de validation traduit une **excellente stabilité** du modèle et une forte capacité de généralisation. La couche LSTM a permis de capturer efficacement les **dépendances temporelles** dans les séquences d'événements issues des traces de mémoire, renforçant la robustesse du modèle face aux variations de comportement entre les logiciels malveillants et les exécutables bénins.

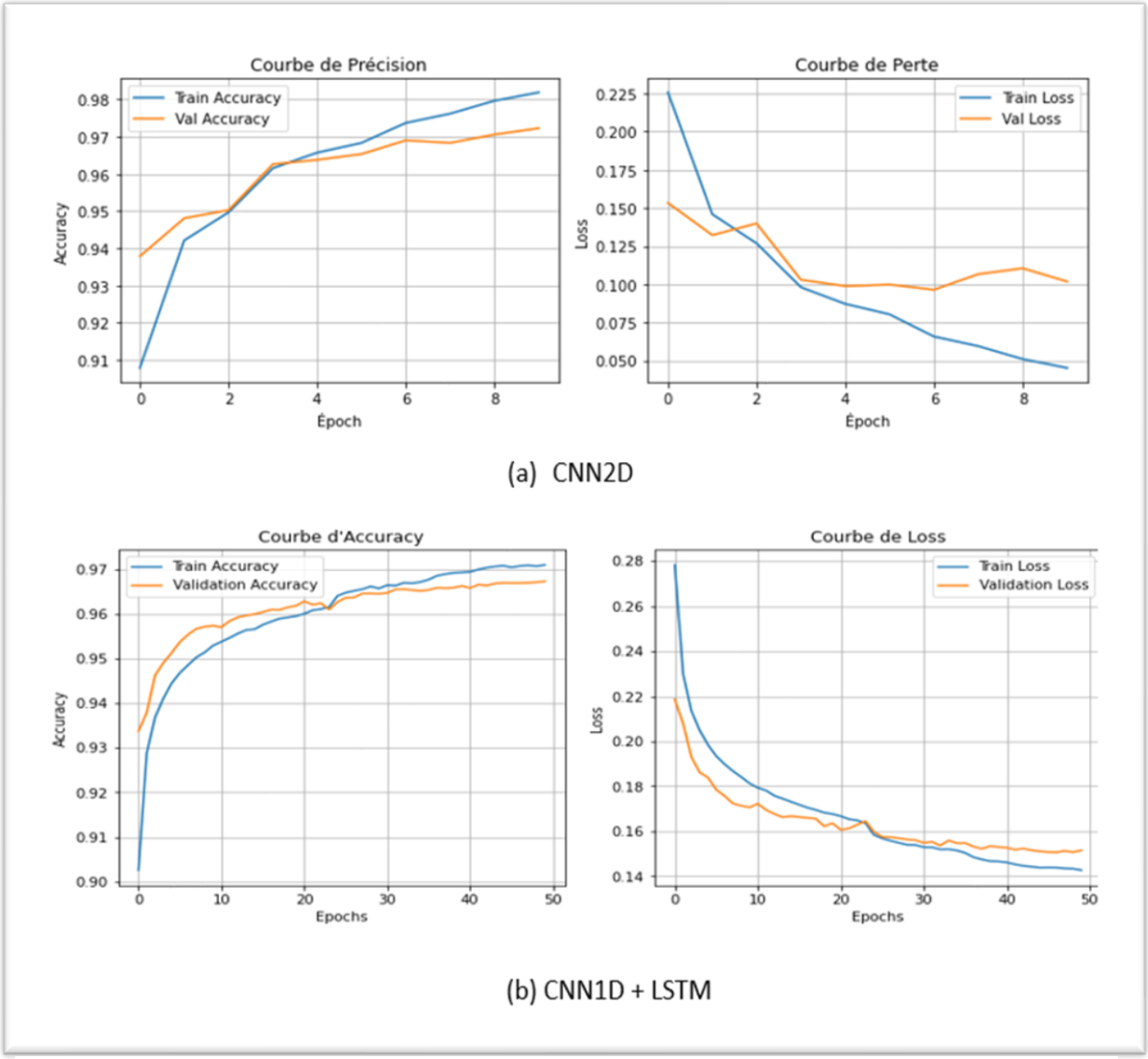


Figure 3- 5: l'évolution de la précision (Accuracy) et de la fonction de perte (Loss) pour les ensembles d'entraînement et de validation au fil des époques sur le dataset Ember

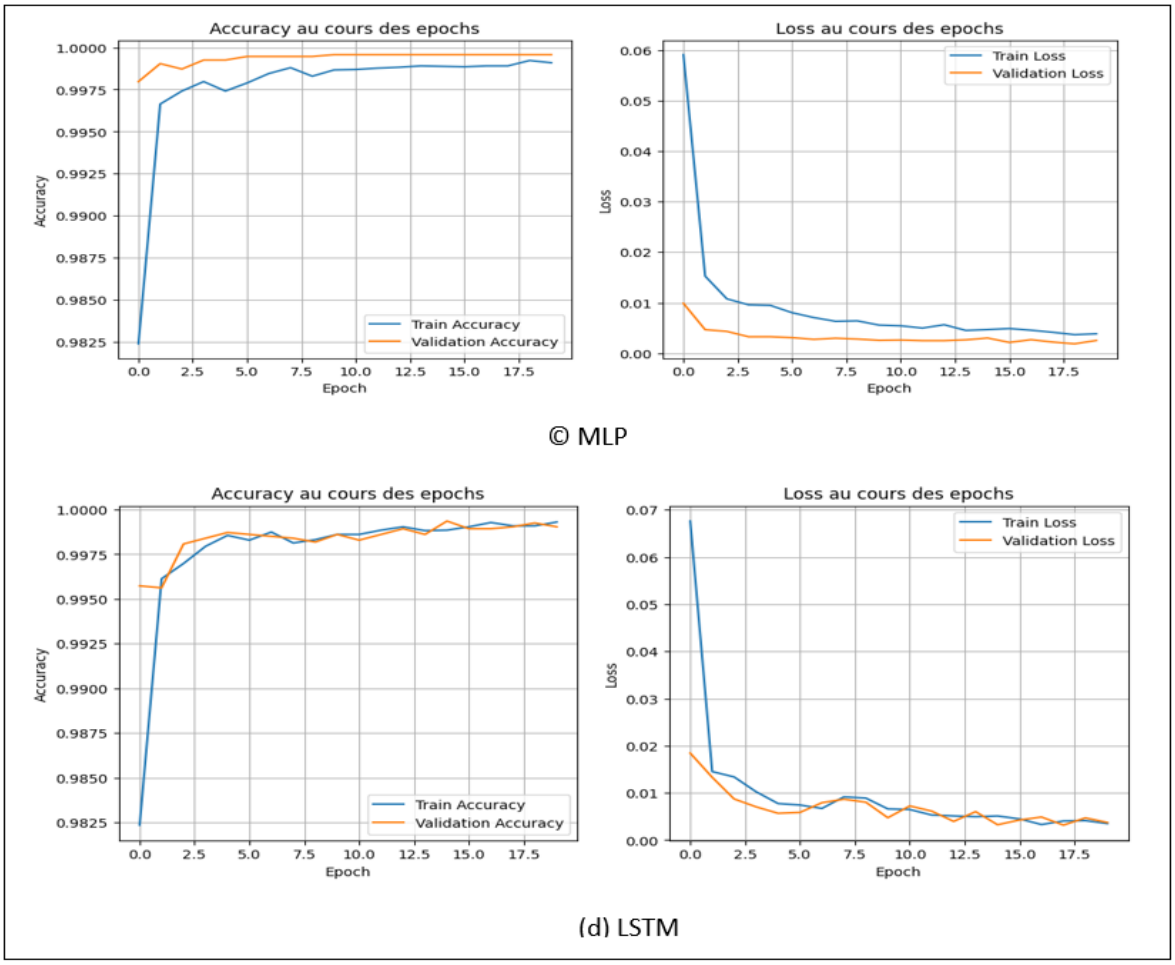


Figure 3- 6: : L'évolution de l'exactitude et la perte des modèles proposés par rapport aux époques d'apprentissage et de validation sur le dataset CIC-Malmem2022

3.6.3. Comparaison et analyse globale des performances des modèles

Afin d'évaluer de manière complète l'efficacité des différents modèles de détection développés au cours de ce travail, une comparaison a été effectuée entre les architectures **CNN2D**, **CNN1D + LSTM**, **MLP**, et **LSTM**, sur les deux jeux de données respectifs **EMBER** et **CIC-MalMem2022**.

Les performances des modèles ont été mesurées principalement à travers les indicateurs suivants : **Accuracy (taux de précision)**, **Loss (fonction de perte)**, ainsi que l'analyse visuelle des courbes d'évolution au cours de l'entraînement et de la validation. La comparaison des performances des modèles est illustrée dans le tableau3-6 suivant :

Tableau3- 6 : Résumé comparatif des performances des modèles

Modèle	Dataset	Accuraccy Entrainement	Accuraccy Validation	Loss Entrainement	Loss Validation
CNN2D	EMBER	0.981	0.972	0.045	0.10
CNN1D+ LSTM	EMBER	0.973	0.968	0.14	0.15
MLP	CIC-Malmem2022	0.999	0.999	< 0.01	< 0.01
LSTM	CIC-Malmem2022	0.998	0.999	< 0.01	< 0.01

3.7. Mesures d'évaluation des modèles

L'évaluation d'un modèle d'apprentissage automatique constitue une étape importante dans tout processus de modélisation. Elle permet de mesurer la capacité du modèle à généraliser sur des données inédites, c'est-à-dire à bien prédire les échantillons qu'il n'a jamais vus pendant l'entraînement. Dans le cadre de la détection de malwares à partir des logs système, plusieurs métriques sont utilisées pour juger la qualité des prédictions notamment.

- **La matrice de confusion** (ou matrice d'erreur) est une méthode qui permet de visualiser les résultats d'un algorithme de classification. Plus précisément, il s'agit d'une table qui décompose le nombre d'instances de vérité terrain d'une classe donnée par rapport au nombre d'instances de classes prédites. Elle est l'un des nombreux indicateurs qui permettent d'évaluer la performance des modèles de classification. On peut s'en servir pour calculer un certain nombre d'autres indicateurs de performance des modèles, tels que la précision et le rappel. En considérant notre cas de figure, qui est la classification binaire, la matrice de confusion est généralement présentée sous forme de matrice 2x2. Il y'a quatre éléments clés dans cette matrice : les vrais positifs (**TP**), les vrais négatifs (**TN**), les faux positifs (**FP**), et les faux négatifs (**FN**).

Voici un exemple de matrice de confusion binaire représenté dans la figure3-7 :

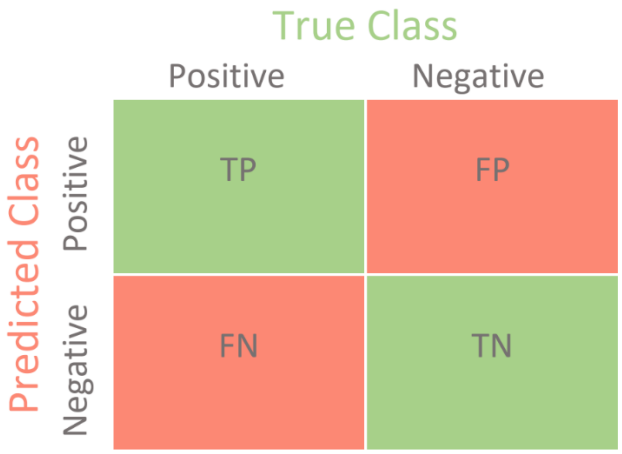


Figure 3- 7: Illustration d'une matrice de confusion [W14]

Interprétation des indicateurs :

- **Vrai Positif (TP)** correspond à un cas ou le modèle prédit correctement la présence d'un malware. Autrement dit, l'échantillon est effectivement malveillant dans la réalité, et le modèle le reconnaît comme tel.
- **Vrai Négatif (TN)** représente un cas où le modèle prédit correctement l'absence de malware. C'est-à-dire que le fichier ou le processus est bénin et que le modèle le classe correctement comme bénin.
- **Faux Positif (FP**) désigne un cas où le modèle classifie à tort un fichier bénin comme un malware. Ce type d'erreur est problématique car il entraîne des fausses alertes. Un taux élevé de FP peut générer une surcharge d'alertes inutiles et nuire à la confiance du modèle.
- **Faux Négatif (FN)** se produit lorsque le modèle n'identifie pas un malware réel**,** le classant à tort comme bénin. C'est le type d'erreur le plus critique en cybersécurité, car cela signifie qu'une menace passe inaperçue et peut compromettre le système. L'objectif essentiel du modèle est donc de minimiser le nombre de FN, même si cela peut parfois augmenter légèrement les FP.
- La **précision** mesure la proportion d'échantillons identifiés comme malwares qui sont réellement malveillants. Une précision élevée signifie que le modèle commet peu de fausses alertes. Elle est définie comme suit :

$$P_r = \frac{TP}{TP + FP} \quad (2)$$

- Le **rappel (Recall)** indique la capacité du modèle à identifier correctement tous les malwares présents dans le jeu de données. Un rappel élevé montre que le modèle parvient à détecter la majorité des menaces, même si cela se fait parfois au prix d'un nombre plus élevé de fausses alertes.

$$Recall = \frac{TP}{TP + FN} \quad (3)$$

- Le **F1-score** est la moyenne harmonique entre la précision et le rappel. Il offre un compromis entre ces deux mesures :

$$F1_{_score} = 2x \frac{P_r \ x \ Recall}{P_r + Recall} \quad (4)$$

- Le **taux de détection (TPR)**, également appelé **sensibilité**, mesure la proportion de malwares effectivement détectés par le modèle parmi tous les malwares présents dans le jeu de données. Il est défini comme suit :

$$TPR = \sum \frac{TP}{Malware} \quad (5)$$

- Le **taux de fausses alertes (FPR)** mesure la proportion de fichiers bénins que le modèle classe à tort comme malwares. Il est défini par la relation suivante :

$$FPR = \sum \frac{FP}{Benin} \quad (6)$$

- **Aire sous la courbe ROC (AUC-ROC)** : La courbe des caractéristiques de fonctionnement du récepteur ROC ou Receiver Operating Characteristic et sa métrique associée, l'aire sous la courbe AUC (Area Under Curve), fournissent une visualisation et une évaluation complètes des performances d'un modèle sur différents seuils. La courbe ROC représente le taux de vrais positifs (sensibilité) par rapport au taux de faux positifs à différents réglages de seuil. Chaque point de la courbe correspond à un seuil spécifique, reflétant

l'arbitrage entre vrais positifs et faux positifs. Plus la valeur de l'AUC est proche de 1, meilleur est le modèle.

3.8. Evaluation sur l'ensemble des données Test

Après la phase d'entrainement, les modèles seront évalués sur l'ensemble de test à l'aide des mesures d'évaluation présentées précédemment. L'ensemble des données de test permet donc de juger la fiabilité du modèle dans un contexte réel**,** en simulant la détection de nouveaux fichiers malveillants ou bénins jamais rencontrés auparavant. Les résultats des mesures d'évaluations obtenus sur l'évaluation des données de test sont représentés dans les figures suivantes.

3.8.1 Evaluation des modèles à partir des métriques

La figure suivante montre la comparaison des performances des quatre modèles sur les ensembles de test des jeux de données EMBER et CIC-Malmem2022 à partir des métriques de classification usuelles.

	precision	recall	f1-score	support
Benign	0.95	0.94	0.94	100000
Malware	0.94	0.95	0.95	100000
accuracy			0.94	200000
macro avg	0.94	0.94	0.94	200000
weighted avg	0.94	0.94	0.94	200000

(a) CNN2D

	precision	recall	f1-score	support
Benign	0.84	0.79	0.81	100000
Malware	0.80	0.85	0.83	100000
accuracy			0.82	200000
macro avg	0.82	0.82	0.82	200000
weighted avg	0.82	0.82	0.82	200000

(b) CNN1D + LSTM

	precision	recall	f1-score	support
Benign	1.00	1.00	1.00	5860
Malware	1.00	1.00	1.00	5860
accuracy			1.00	11720
macro avg	1.00	1.00	1.00	11720
weighted avg	1.00	1.00	1.00	11720

(c) MLP

	precision	recall	f1-score	support
Benign	1.00	1.00	1.00	5860
Malware	1.00	1.00	1.00	5860
accuracy			1.00	11720
macro avg	1.00	1.00	1.00	11720
weighted avg	1.00	1.00	1.00	11720

(d) LSTM

Figure 3- 8: Évaluation comparative des modèles CNN2D, CNN1D+LSTM, MLP et LSTM sur les ensembles de test des jeux de données EMBER et CIC-Malmem2022 pour les métriques : précision, rappel, F1-score et exactitude.

3.8.2 Matrice de confusion

Afin d'analyser les performances de classification des différents modèles d'apprentissage, la figure3-9 ci-dessous illustre les matrices de confusion correspondantes. Elles mettent en évidence la répartition des prédictions correctes (vrais positifs et vrais négatifs) et des erreurs (faux positifs et faux négatifs) pour chaque architecture de modèles implémentés.

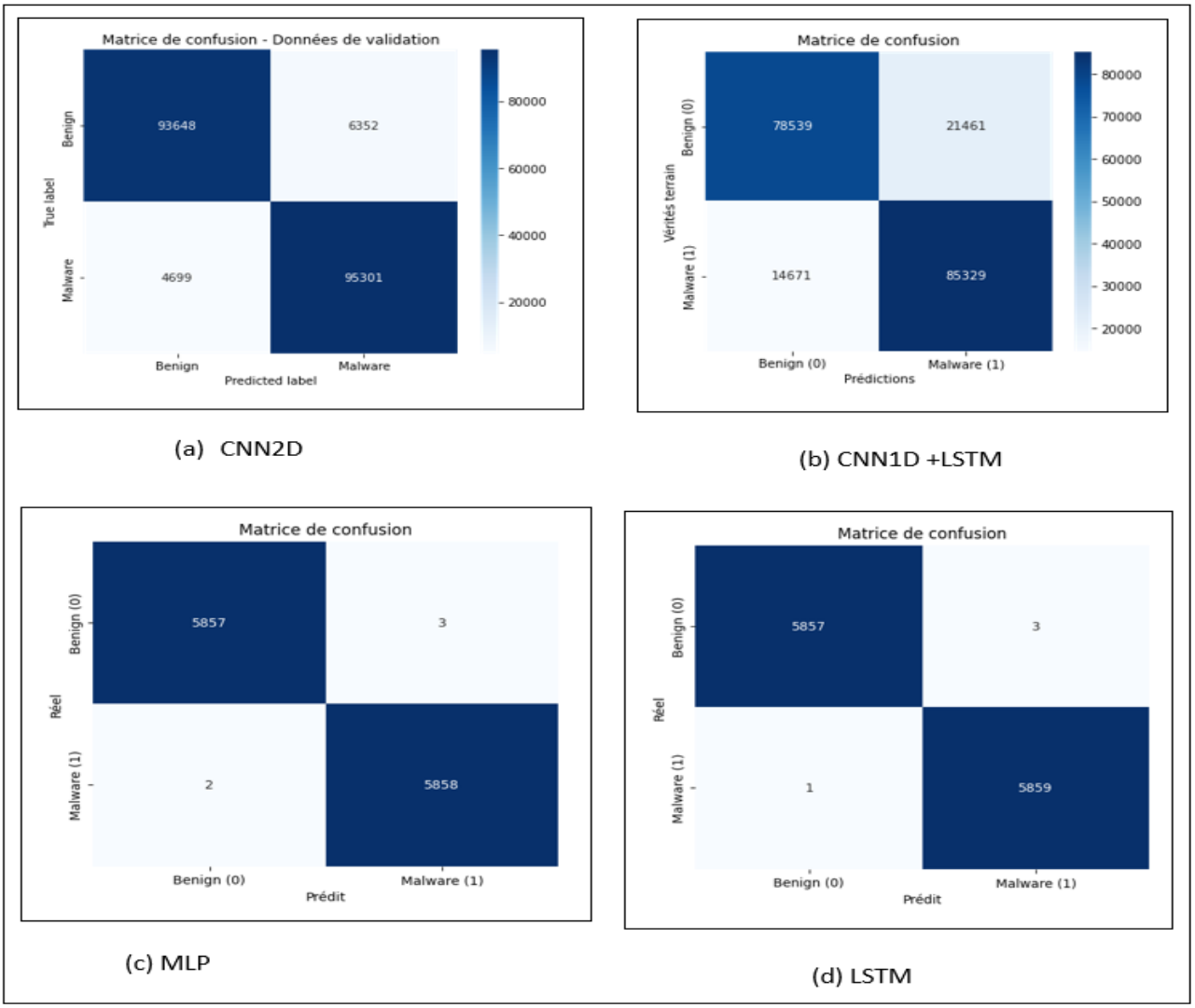


Figure 3- 9: Matrices de confusions des modèles CNN2D, CNN1D+LSTM, MLP et LSTM sur les ensembles de test des deux jeux de données (ember et cic-malmem2022).

3.8.3 La courbe Receiver Operating Characteristic (ROC)

La figure3-10 suivante illustre la performance de quatre modèles de détection de malwares à travers leurs courbes ROC et les valeurs correspondantes de l'AUC (Area Under the Curve). Les sous-figures (a) à (d) représentent respectivement : (a)CNN2D, (b)CNN1D + LSTM, (c)MLP, et (d)LSTM. Ces courbes illustrent la capacité de chaque modèle à distinguer les malwares des logiciels bénins, un AUC proche de 1 indique une très bonne performance.

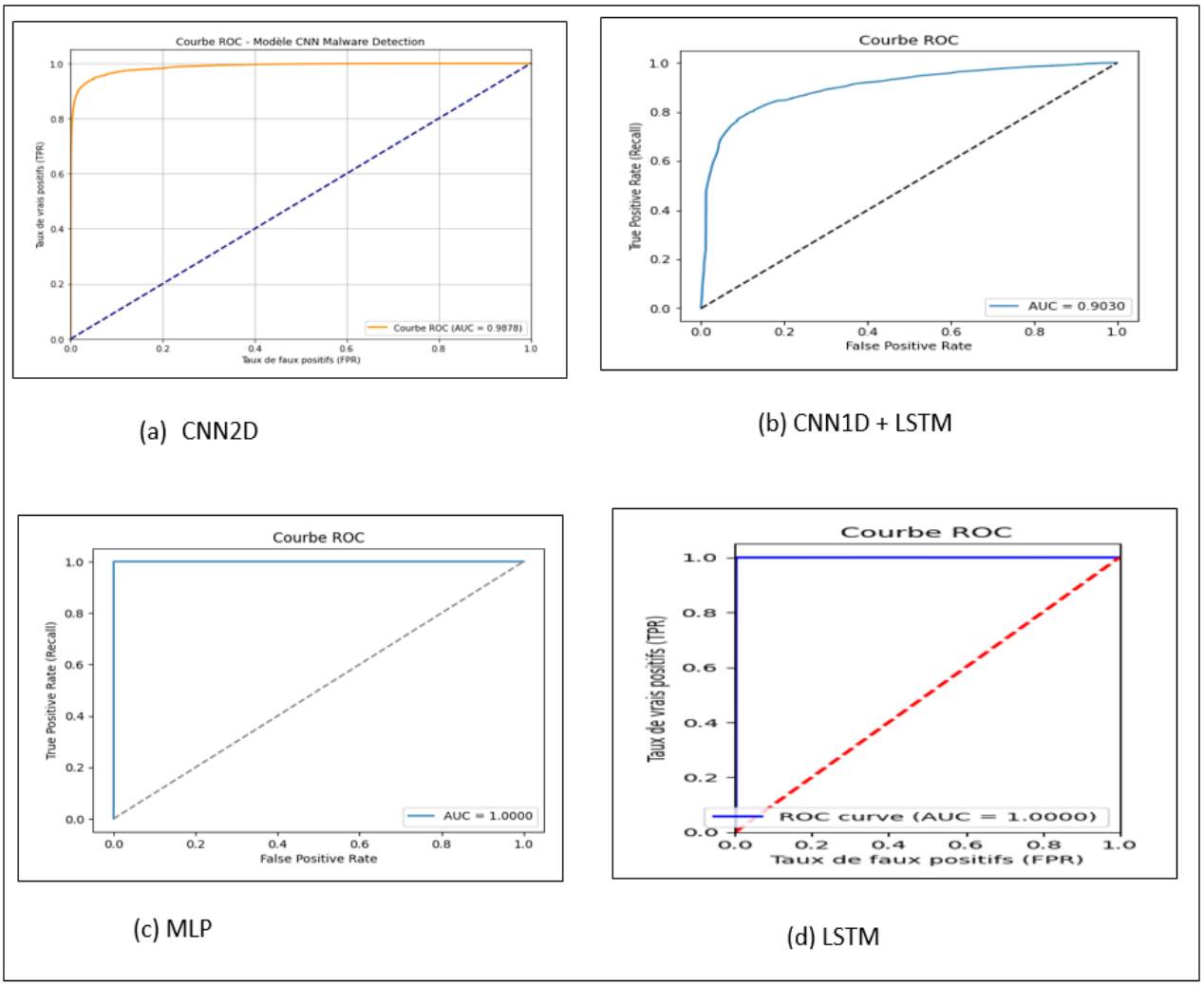


Figure 3- 10: Analyse comparative des performances des modèles de détection de malwares à partir des courbes ROC

3.9. Interprétation générale des résultats

L'analyse croisé des courbes d'entrainement (Accuracy et Loss) et des métriques d'évaluation (matrice de confusion, F1-Score, Recall, ROC, etc.) des modèles de détection de malwares a été conduite sur deux jeux de données distincts : **CIC-Malmem2022** et **EMBER**. Cette approche comparative vise à analyser la performance, la robustesse et la capacité de généralisation de différents modèles d'apprentissage profond face à des caractéristiques de fichiers variés. Les modèles considérés dans cette étude sont le **Perceptron Multicouche (MLP)**, le **réseau de neurones récurrent LSTM**, le **réseau convolutif bidimensionnel (CNN2D)** et le **modèle hybride CNN1D–LSTM**.

3.9.1. Analyse des courbes d'entraînement et de validation : convergence et capacité d'apprentissage

L'analyse des courbes d'accuracy et de loss issues de l'entraînement des différents modèles constitue un indicateur essentiel pour évaluer leur capacité d'apprentissage, leur stabilité et leur aptitude à généraliser sur les données de validation. Les courbes obtenues révèlent des tendances contrastées selon les architectures testées et la nature des données.

3.9.1.1 Résultats sur le dataset EMBER

Compte tenu de la nature très différente du jeu de données ember ; avec des données plus volumineuses, des caractéristiques statiques complexes et une forte variabilité entre fichiers PE. Les figure3-5(a) et figure3-5(b) illustrent clairement cette différence de difficulté.

- **Le modèle CNN2D**

Dans ce modèle on observe :

- Une augmentation progressive de l'accuracy, atteignant un plateau intermédiaire qui reste néanmoins satisfaisant ;
- Une loss, une fonction de perte qui diminue régulièrement ;
- Un écart modéré entre les courbes d'entraînement et de validation, qui reste toutefois maîtrisé.

Le CNN2D parvient à extraire des représentations discriminantes à partir des données malgré leur absence de structure spatiale naturelle. Sa convergence reste stable, ce qui montre que le modèle apprend des motifs pertinents.

L'écart léger mais constant entre entraînement et validation suggère toutefois une limite de généralisation due à la complexité du dataset.

- **Modèle hybride CNN1D + LSTM**

Cette architecture, combinant une extraction spatiale via CNN1D et une modélisation temporelle à travers LSTM, semble moins adaptée au caractère statique et hautement variable des features de EMBER.

L'écart entrainement et validation traduit un manque de stabilité et une généralisation plus faible. Le modèle apprend, mais peine à capturer des motifs suffisamment cohérents pour atteindre des performances élevées. Ce comportement illustre la difficulté de généralisation face à un dataset riche, complexe et hétérogène.

La figure3-5(b) présente :

- Une accuracy qui croît mais atteint un niveau inférieur, révélant une difficulté accrue d'apprentissage ;
- Une loss qui diminue mais reste relativement élevée, sans atteindre le niveau attendu
- Un écart plus visible entre les courbes d'entraînement et de validation.

3.9.1.2 Résultats sur le dataset CIC-MalMem2022

Les résultats des courbes d'entrainement et de validation du jeu de données CIC-MalMem2022 sont illustrés dans les figure3-6© et figure3-6(d). Ci-dessous une analyse des résultats avec les différents modèles étudiés.

- **MLP (Multilayer Perceptron)**

Les courbes d'apprentissage du MLP montrent une évolution très rapide et stable tout au long des époques :

- L'accuracy d'entraînement augmente de manière quasi linéaire dès les premiers epochs, atteignant un niveau extrêmement élevé.
- L'accuracy de validation suit presque exactement la même trajectoire, ce qui est le signe d'une excellente capacité de généralisation.
- La loss d'entraînement diminue rapidement jusqu'à devenir presque nulle.
- La loss de validation suit une évolution parallèle, sans remontée brutale ni oscillations.

Ce comportement démontre que le MLP apprend avec une grande facilité les motifs du dataset. La superposition quasi parfaite entre les courbes d'entraînement et de validation indique que le modèle ne souffre ni d'overfitting ni d'underfitting.

Cela s'explique par la grande séparabilité des classes dans CIC-MalMem2022, où les caractéristiques sont bien structurées et peu bruitées. Le modèle parvient ainsi à construire rapidement une excellente capacité de généralisation.

- **Modèle LSTM**

Les courbes d'entraînement et de validation du LSTM confirment également une convergence très efficace :

- L'accuracy progresse de façon régulière, sans oscillations, jusqu'à atteindre un niveau très élevé.
- La loss décroît également de manière continue et lisse.
- Les courbes train/val restent extrêmement proches tout au long de l'entraînement.

En résumé, Le LSTM, par sa capacité à capturer les dépendances temporelles, exploite très efficacement la structure séquentielle des données. Le comportement très stable des courbes indique un apprentissage sans instabilité et sans surajustement. Cette architecture apparaît parfaitement adaptée au type de données contenues dans CIC-MalMem2022, ce qui justifie ses performances particulièrement élevées.

En conclusion, les modèles simples MLP, LSTM excellent lorsque les données sont bien structurées, tandis que les architectures profondes CNN2D, CNN1D+LSTM s'imposent pour des données plus complexes mais nécessitent un réglage plus poussé. La convergence et la capacité d'apprentissage des modèles dépendent fortement du dataset utilisé.

3.9.2. Interprétation des métriques d'évaluation sur les données test

L'évaluation des modèles a été réalisée en s'appuyant sur plusieurs métriques de classification essentielles : **la précision**, **le rappel**, **le F1-score**, **l'accuracy**, ainsi que l'analyse des matrices de confusion. Ces indicateurs permettent d'apprécier la capacité réelle des modèles à distinguer efficacement les échantillons malveillants et bénins sur l'ensemble de test.

- **Résultat sur le dataset CIC-Malmem2022**

Les résultats des métriques sur les figure3-8(c) et figure 3-8(d) montrent que les modèles MLP et LSTM obtiennent des scores quasi parfaits sur ce dataset. La précision, le rappel et le F1-score dépassent **99,9 %**, traduisant une capacité exceptionnelle à identifier correctement aussi bien les malwares que les échantillons bénins.

Le **rappel très élevé** environ 99,98 % indique que le modèle laisse passer **très peu de malwares** (faible faux négatifs), un critère critique en cybersécurité. De même, la **précision proche de 100 %** reflète une quasi-absence de fausses alertes (faux positifs), ce qui évite de classifier des fichiers légitimes comme malveillants.

Dans ce cas, on peut dire que le comportement des modèles MLP et LSTM sur CIC-MalMem2022 confirme que ce dataset, basé sur des séquences de comportements d'exécution, est particulièrement bien adapté à une architecture récurrente capable de modéliser les dépendances temporelles. Les métriques témoignent d'une **excellente généralisation**, avec des erreurs marginales.

- **Résultats sur le dataset EMBER**

  - **CNN2D**

Le modèle CNN2D applique un traitement spatial sur les vecteurs de 2335 caractéristiques du dataset EMBER. Les métriques obtenues dans la figure3-8(a) montrent des performances nettement élevées, avec :

  - Précision élevée, traduisant une faible proportion de faux positifs,

- Rappel satisfaisant, bien que plusieurs milliers de malwares restent mal classifiés,
- F1-score autour de **95 %**, témoignant d'un bon équilibre entre rappel et précision.

Bien que les performances soient moins élevées que sur CIC-MalMem, le modèle atteint un niveau solide pour un dataset réputé difficile, fortement bruité et contenant beaucoup de variabilité entre échantillons. Le CNN2D constitue une architecture pertinente pour EMBER. Les métriques de classification démontrent sa capacité à extraire des relations complexes entre features et à offrir une détection fiable, même si une marge d'amélioration demeure au niveau du rappel.

- **Modèle hybride CNN à une dimension 1D avec du LSTM**

Dans le cas du modèle hybride CNN1D+LSTM, les métriques de classification de ce modèle montrent une **dégradation notable** des performances par rapport au CNN2D. Le rappel chute significativement, traduisant un nombre élevé de malwares non détectés. La précision reste correcte mais inférieur à celle du CNN2D, et le F1-score est sensiblement moins élevé, indiquant une perte d'équilibre entre la capacité à détecter les malwares et celle à limiter les fausses alertes.

Constat : Le modèle éprouve des difficultés à capter les relations pertinentes à partir des vecteurs EMBER, malgré l'ajout d'une couche séquentielle LSTM censée améliorer la modélisation séquentielle. Cela suggère que la transformation des vecteurs en séquences et l'application d'un LSTM n'apporte pas d'avantage avec ce type de données.

Comparaison des deux datasets sur les métriques d'évaluation :

D'après l'analyse des résultats obtenus des métriques d'évaluation avec les deux jeux de données ; on constate que :

- La nature du dataset influence fortement les résultats,
- Les modèles doivent être adaptés à la structure des données,

- Les métriques de classification confirment la supériorité des modèles MLP et LSTM sur l'ensemble des données CIC-MalMem,
- Le CNN2D est plus performant que le modèle hybride sur le dataset ember,
- Le modèle hybride présente un compromis insuffisant pour une détection fiable.

3.9.3. Interprétation des matrices de confusion et des courbes ROC

L'analyse conjointe des **matrices de confusion** et des **courbes ROC** permet d'obtenir une vision complète de la performance des modèles sur les deux jeux de données. Alors que les matrices de confusion permettent d'apprécier la répartition exacte des erreurs (faux positifs et faux négatifs), les courbes ROC et leurs scores AUC offrent une mesure globale sur la capacité discriminante du classificateur.

Sur le dataset **CIC-Malmem2022**, les modèles **MLP** et **LSTM** ont obtenu des résultats particulièrement satisfaisants, comme l'illustrent les matrices de confusion présentées dans les figure3-9(c) et figure3-9(d). Ces résultats montrent une quasi-parfaite séparation entre les classes bénignes et malveillantes, avec seulement quelques erreurs de classification (faux positifs et faux négatifs inférieurs à 5) sur plus de 11 700 échantillons. Les faux positifs et faux négatifs sont extrêmement rares, ce qui se traduit par une précision, un rappel et un F1-score proches de **100 %**.

Les courbes ROC dans les figure3-10(c) et figure3-10(d) confirment ces résultats : elles épousent presque parfaitement le coin supérieur gauche du graphique, ce qui se traduit par un **AUC ≈ 1.0**. Cela signifie que le modèle maintient un taux de vrais positifs extrêmement élevé même pour un très faible taux de faux positifs.

Les modèles **CNN2D** et **CNN1D–LSTM** ont été évalués sur le dataset **EMBER**, beaucoup plus vaste et complexe. Les matrices de confusion obtenues sont représentées dans les figures suivantes figure3-9(a) et figure3-9(b).

Le modèle **CNN2D** affiche de bonnes performances globales, avec une **AUC de 0.9878** dans la figure3-10(a) et un équilibre satisfaisant entre les taux de détection et de fausses alertes avec un nombre non négligeable de **faux positifs** (**6 352**), ainsi que des **faux négatifs** (**4 699**).

Ces résultats confirment sa **capacité à extraire des motifs complexes** à partir de données à forte dimensionnalité de 2335 caractéristiques.

En revanche, le **modèle hybride CNN1D–LSTM** présente des performances légèrement inférieures, marquées par un taux d'erreur plus élevé. La matrice de confusion du modèle hybride révèle davantage difficultés, avec un volume plus important de mauvaises classifications, notamment des faux négatifs (malwares non détectés) et des faux positifs.

La courbe ROC associée figure3-10(b) est plus aplati que celle du CNN2D, avec un AUC inférieur, traduisant une capacité discriminante moins robuste. Ce comportement peut être attribué à la complexité du dataset EMBER et à la sensibilité de l'architecture hybride au choix des hyperparamètres (nombre de filtres, taille du noyau, taux de dropout, etc.).

Les matrices de confusion et les courbes ROC confirment que le LSTM est le modèle le plus performant sur CIC-MalMem2022, bénéficiant d'un dataset très bien séparé. Et sur EMBER, le CNN2D est le modèle le plus robuste, avec une bonne capacité discriminante malgré la complexité des données. Alors que le modèle hybride CNN1D+LSTM est le moins performant sur EMBER, ce que révèlent à la fois les erreurs observées et la faiblesse de la courbe ROC.

3.9.4. Tableau comparatif final des résultats d'évaluation

Afin de synthétiser l'ensemble des résultats obtenus lors de l'évaluation des modèles, le tableau3-7 ci-dessous présente une comparaison globale des performances pour les deux jeux de données utilisés. Il regroupe les principales métriques de classification (précision, rappel, F1-score, etc), ainsi que les AUC, permettant ainsi d'apprécier, de manière structurée et cohérente le comportement de chaque modèle face aux caractéristiques propres à chaque dataset. Cette présentation comparative facilite l'interprétation des performances et met en évidence les modèles les plus efficaces selon la nature des données analysées.

Tableau3- 7 : Tableau comparatif final des résultats d'évaluation

Dataset	Modèle	Accuracy	Précision	Recall (TPR)	F1-score	ROC-AUC	Faux positifs	Faux négatifs
CIC-Malmen 2022	MLP	100%	100%	100%	~100 %	≈1.00	3	2
	LSTM	100%	100%	100%	100%	≈1.00	3	1
EMBER	CNN2D	Elevée	Elevée	Bonne	Bonne	Haute	6 352	4 699
	CNN1D +LSTM	Moyenne	Moyenne	Moyenne	Moyenne	Moyenne	21 461	14 671

L'analyse comparative met en évidence que les performances des modèles varient fortement selon la nature du dataset. Sur CIC-MalMem2022, les modèles classiques tels que le **MLP** et le **LSTM** obtiennent des résultats presque parfaits, traduisant une séparation claire entre les classes. D'un autre côté, sur le dataset, plus complexe et plus hétérogène qui est **EMBER**, les modèles profonds comme le **CNN2D** et le **CNN1D+LSTM** montrent une meilleure capacité de généralisation malgré une baisse globale des métriques. Ces résultats soulignent l'importance du choix du modèle en fonction des caractéristiques structurelles du dataset et confirment la nécessité d'approches plus profondes pour les jeux de données à forte variabilité.

3.10. Discussion avec les travaux connexes

Comparaison de l'étude avec la revue de littérature

Les résultats obtenus dans ce travail se positionnent favorablement par rapport aux études existantes présentées dans la littérature. Sur le dataset EMBER, les modèles CNN2D et CNN1D+LSTM ont atteint une accuracy respective de 98,1 % et 97,3% en entrainement, ce qui est comparable aux performances rapportées par Shah et Nawaf (2024) avec des architectures CNN1D et hybrides CNN–LSTM (≈ 96% et 97%), et supérieur aux résultats obtenus dans certains travaux basés sur des CNN classiques ou 1D, tels que ceux de Leveau et

al. (2021) ou Marais (2023). Ces résultats confirment l'efficacité des représentations bidimensionnelles et des réseaux convolutionnels profonds pour l'analyse statique des fichiers malveillants.

Sur le dataset CIC-MalMem2022, les modèles MLP et LSTM atteignent des performances très élevées, avec une accuracy de 99,9 % et une perte inférieure à 0,01, ce qui est cohérent avec les travaux basés sur l'analyse mémoire tels que Orman et al. (2022) et MeMalDet et al. (2024), qui rapportent également des scores supérieurs à 98 %. Contrairement à certaines études où les modèles classiques surpassent le Deep Learning, les résultats obtenus montrent qu'une architecture profonde adaptée aux données peut rivaliser avec les meilleures approches de l'état de l'art.

Dans l'ensemble, ces comparaisons indiquent que les performances obtenues sont en adéquation avec les meilleurs résultats rapportés dans la littérature récente, tout en reposant sur des architectures relativement simples.

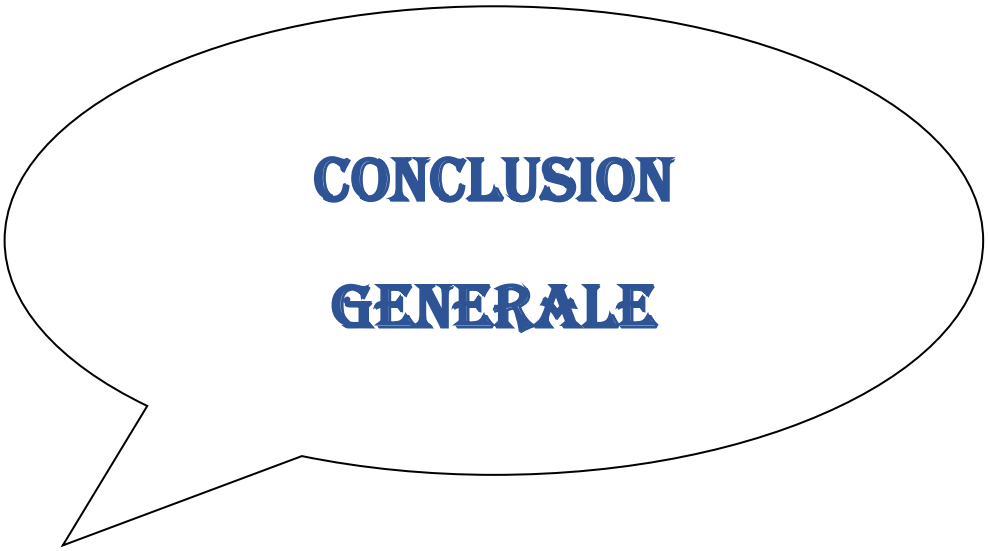
3.11. Conclusion

Ce chapitre a permis de présenter en détail le processus complet de conception, d'implémentation et d'évaluation des différents modèles de détection de malwares développés dans le cadre de ce travail. Après une phase de préparation et de structuration des données, plusieurs architectures d'apprentissage profond notamment le MLP, le LSTM**,** le CNN2D et le modèle hybride CNN1D-LSTM ont été mises en œuvre afin d'exploiter au mieux les caractéristiques spécifiques des datasets CIC-MalMem2022 et EMBER.

Les résultats obtenus lors de l'entraînement, illustrés par les courbes d'accuracy et de loss, ont permis d'apprécier les dynamiques d'apprentissage propres à chaque modèle. L'évaluation sur les données de test, réalisée à travers les matrices de confusion, les métriques de classification et les courbes ROC, a montré des performances contrastées : des résultats presque parfaits sur CIC-MalMem2022 et des performances plus modérées sur EMBER, dont la complexité et la forte variabilité nécessitent des modèles plus profonds et plus robustes.

Il convient toutefois de souligner que ce processus de réalisation n'a pas été exempt de difficultés. L'entraînement sur le dataset EMBER a notamment été marqué par des

contraintes computationnelles importantes, liées au volume élevé des données et à la dimensionnalité des features. Certains modèles ont également présenté des tendances au **surapprentissage**, nécessitant l'ajustement de plusieurs hyperparamètres (taux d'apprentissage, dropout, taille des batchs). Ces limites ont influencé les choix méthodologiques sur le dataset ember.


CONCLUSION
GENERALE

Conclusion Générale

La prolifération des logiciels malveillants et l'évolution croissante de leurs techniques d'évasion constituent aujourd'hui un enjeu majeur pour la cybersécurité. Face aux limites des approches traditionnelles basées sur les signatures ou sur l'analyse statique, les techniques d'apprentissage profond se présentent comme des solutions prometteuses, capables d'extraire automatiquement des caractéristiques complexes et de détecter des comportements malveillants difficiles à identifier par des méthodes classiques. Ce mémoire s'est inscrit dans cette dynamique en explorant la conception, l'implémentation et l'évaluation de modèles de Deep Learning appliqués à la détection des malwares.

L'étude de ce travail est répartie en trois grands chapitres dont le premier chapitre « Etat de l'art sur la détection des malwares » a permis de positionner scientifiquement le domaine de la détection des malwares et de mettre en évidence les limites inhérentes aux solutions traditionnelles. Dans la deuxième partie sur les fondements théoriques des modèles de Deep Learning utilisés dans le cadre de ce projet. Les architectures des modèles ont été décrits en détail afin de mettre en évidence leur pertinence pour la détection des logiciels malveillants.

La troisième partie a porté sur la conception et la réalisation des modèles développés. Après le choix des ensembles de données EMBER et CIC-MalMem2022 et leurs prétraitements, plusieurs modèles comme le CNN2D, le modèle hybride CNN1D+LSTM, MLP et LSTM ont été entraînés et testés. Les résultats montrent des performances remarquables sur le dataset CIC-MalMem2022, avec des métriques quasiment parfaites sur les modèles évalués. À l'inverse, les performances obtenues sur EMBER, bien que solides, révèlent une complexité nettement plus élevée, que seuls les modèles les plus profonds arrivent à donner des performances acceptables. Les courbes d'apprentissage, les matrices de confusion et les courbes ROC ont permis d'effectuer une analyse fine des modèles et de comprendre leurs limites, notamment les risques de surapprentissage et les contraintes liées aux performances du système rencontrées durant l'entraînement.

Globalement, les travaux réalisés dans ce mémoire démontrent l'impact réel du Deep Learning pour la détection avancée des malwares. Toutefois, certaines limites demeurent, qu'il s'agisse de la variabilité des formats de données, de la nécessité de traitements préalables plus robustes ou encore des besoins en ressources de calcul. Ces aspects ouvrent des pistes pour des travaux futurs, tels que l'exploration de modèles plus légers optimisés pour les environnements embarqués, l'utilisation de techniques d'apprentissage

auto-supervisé ou encore l'intégration de données dynamiques issues de l'analyse comportementale.

En définitive, ce mémoire contribue à la compréhension et à l'amélioration des techniques de détection des malwares via l'apprentissage automatique et profonde, tout en posant les bases pour des recherches futures visant à renforcer la résilience des systèmes face aux menaces émergentes.

Perspectives :

Dans nos travaux futurs, nous allons améliorer ce travail sur deux parties : l'**enrichissement des données et** l'utilisation de **méthodes d'apprentissage avancées**.

Les modèles utilisés dans ce mémoire reposent sur des données statiques et dynamiques prises de manière séparée. L'intégration de données hybrides (statique et dynamiques) pourrait améliorer significativement la robustesse et la capacité de détection des modèles face aux malwares obfusqués.

L'utilisation de **méthodes d'apprentissage avancées**, telles que l'apprentissage auto-supervisé, l'apprentissage contrastif ou les modèles génératifs (GANs, VAEs), ouvre des possibilités intéressantes pour réduire la dépendance aux données étiquetées, souvent coûteuses et difficiles à obtenir dans le domaine des malwares.

BIBLIOGRAPHIE:

[1] A. H. Thiziers, K. T. Tiémoman, N. B. Gérard, et T. Q. K. Traoré, "Malware Detection Using Deep Learning," *Open Journal of Applied Sciences, 2023, 13, 2480-2491*, 29 déc. 2023. [En ligne, disponible: https://www.scirp.org/journal/ojapps]

[2] A. Chérifi, *Un Framework de sécurité pour détecter les malwares Android*, Mémoire de Master, Université de 8 Mai 1945 Guelma, 2021 /CHERIFI_ABDERRAOUF_F5_Informatique

[3] Auray, N. L'invisible et le clandestin. Une ethnographie du virus informatique Storm. Terrain. Anthropologie & sciences humaines, (2015), (64), 32-49

[4] Filiol, E. (2009). ´ Les virus informatiques : théorie, pratique et applications. Springer Science & Business Media

[5] Cédric Herzog. Etude des malwares évasifs : conception, protection et détection. Cryptographie et sécurité [cs.CR]. CentraleSupélec, 2022. Français. : 2022CSUP0001ff. fftel-04003402

[6] K. Brezinski and K. Ferens, "Metamorphic Malware and Obfuscation : A Survey of Techniques, Variants, and Generation Kits," *Security and Communication Networks*, vol. 2023, Article ID 8227751, 41 pages, 2023. doi: 10.1155/2023/822775

[7] M. Ijaz, M. H. Durad, and M. Ismail, "Static and Dynamic Malware Analysis Using Machine Learning," in Proceedings of the 16th International Bhurban Conference on Applied Sciences and Technology (IBCAST), Islamabad, Pakistan, Jan. 2019. doi: 10.1109/IBCAST.2019.8667136

[8] D. Gibert, C. Mateu, and J. Planes, "The rise of machine learning for detection and classification of malware: Research developments, trends and challenges," *Journal of Network and Computer Applications*, vol. 153, pp. 102526, Jan. 2020. https://doi.org/10.1016/j.jnca.2019.102526

[9] H. Djallel, "Un système de détection d'intrusion pour la cybersécurité," Mémoire de fin d'études, Université 8 Mai 1945 – Guelma, Oct. 2020.

[10] Robin Sommer and Vern Paxson. Outside the closed world: On using machine learning for network intrusion detection. In 2010 IEEE symposium on security and privacy, pages 305–316. IEEE, 2010.

[11] Li Deng. A tutorial survey of architectures, algorithms, and applications for Deep Learning. APSIPA Transactions on Signal and Information Processing, 3, 2014

[12] Weibo Liu, Zidong Wang, Xiaohui Liu, Nianyin Zeng, Yurong Liu, and Fuad E Alsaadi. A survey of deep neural network architectures and their applications. Neurocomputing, 234 :11–26, 2017

[13] Jiuxiang Gu, Zhenhua Wang, Jason Kuen, Lianyang Ma, Amir Shahroudy, Bing Shuai, Ting Liu, Xingxing Wang, Gang Wang, Jianfei Cai, et al. Recent advances in convolutional neural networks. Pattern Recognition, 77 :354–377, 2018.

[14] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In Advances in neural information processing systems, pages 1097–1105, 2012

[15] Clément Dalloux, Natalia Grabar, and Vincent Claveau. Détection de la négation : corpus français et apprentissage supervisé. Revue des Sciences et Technologies de l'Information-Série TSI : Technique et Science Informatiques, pages 1–21, 2019.

[16] R. Leveau and J.-S. Lerat, "Analyse de malwares et classification à l'aide de réseaux de neurones convolutionnels 1D," in *6ème Journée des Chercheurs en Haute École*, 25 Feb. 2021.

[17] B. Marais, "Améliorations des outils de détection de malwares par analyse statique grâce à des mécanismes d'intelligence artificielle," Thèse de doctorat, Normandie Université, France, 2023. [En ligne]. Disponible sur: https://theses.hal.science/tel-04416984v1

[18] E. Baghirov and M. Afzal, "Unveiling Malicious Patterns: Autoencoder-Based Malware Detection," "System Informaticsé", no. 24, p. 113, 2024.

[19] Halim M.A., Abdullah A., Zainol Ariffin K.A. Recurrent neural network for malware detection // International Journal of Advances in Soft Computing and its Applications. 2019. Vol. 11, No. 1. P. 46– 63.

[20] Panchagnula V.M., N.V.L. Satya Keerthi Ch., Surekha S., Sujatha R., Veeraiah D., Ramesh E., Lakshmi B. A deep learning approach for detecting malware using autoencoder // IAENG International Journal of Computer Science. 2024. Vol. 51, No. 8. P. 1051–1059

[21] Zhang, Z. Prédiction des logiciels malveillants Microsoft à l'aide du modèle LightGBM. 3e Conférence internationale 2022 sur le Big Data, l'intelligence artificielle et l'ingénierie de l'Internet des objets (ICBAIE), Xi'an, du 15 au 17 juillet 2022, 41-44. [En ligne]. Disponible sur : https://doi.org/10.1109/ICBAIE56435.2022.9985850

[22] Huo, D., Li, X., Li, L., Gao, Y., Li, X. et Yuan, J. (2022) L'application de 1D-CNN dans la détection des logiciels malveillants Microsoft. 7e Conférence internationale sur l'analyse des mégadonnées (ICBDA) 2022, Guangzhou, 4-6 mars 2022, 181-187. [En ligne]. Disponible sur : https://doi.org/10.1109/ICBDA55095.2022.9760349

[23] Maniriho P., Mahmood A.N., Chowdhury M.J.M. MeMalDet: A memory analysis-based malware detection framework using deep autoencoders and stacked ensemble under temporal evaluations // Computers & Security. 2024. Vol. 142. DOI: 10.1016/j.cose.2024.103864.

[24] Shah, A., & Nawaf, L. (2024). *Malware Detection Using Deep Learning Approaches*. Preprints. https://doi.org/10.20944/preprints202407.1214.v1

[25] A. Orman *et al.*, “Malware Detection Using Memory Analysis Data in Big Data Environment,” *Applied Sciences*, vol. 12, no. 17, p. 8604, 2022, doi: 10.3390/app12178604

[26] V. M. Panchagnula, C. Satya Keerthi N. V. L., S. Surekha, R. Sujatha, D. Veeraiah, E. Ramesh, and B. Lakshmi, “A Deep Learning Approach for Detecting Malware Using Autoencoder,” *vol.* 51, *no.* 8, pp. 1051–1059, Aug. 2024.

WEBOGRAPHIE

[W1] : Fichiers malveillants sur Kaspersky https://www.kaspersky.fr/about/press-releases/en-2024-kaspersky-a-detecte-467-000-fichiers-malveillants-par-jour , [En ligne ; consulté le 05/03/2025]

[W2] : Violation de données personnelles de l'opérateur Free : situation, risques et recommandations, https://www.cybermalveillance.gouv.fr/tous-nos-contenus/actualites/violation-de-donnees-personnelles-free-202410 , [En ligne ; consulté le 05/03/2025]

[W3]: Logiciels malveillants sur AV-Test https://www.av-test.org/fr/statistiques/logiciels-malveillants/ ; [En ligne ; consulté le 06/03/2025]

[W4] : Protection contre les malwares ajoutée au stockage de marque Paubox https://www.paubox.com/blog/malware-protection-added-to-paubox-branded-storage [En ligne ; consulté le 07/03/2025]

[W5] : Qu'est-ce que l'analyse des logiciels malveillants https://wardeners.com/analyse-malw.html [En ligne ; consulté le 09/03/2025]

[W6]: Analyse statique et analyse dynamique https://fr.parasoft.com/blog/static-analysis-and-dynamic-analysis/ [En ligne; consulté le 10/03/2025]

[W7] : La structure de l'architecture du deep learning https://developer.ibm.com/developer/default/articles/cc-machine-learning-deep-learning-architectures/images/DL-arch.png; [En ligne, consulté le 01/04/2025]

[W8] : Guide ultime des réseaux CNN, https://www.ia-insights.fr/comprendre-les-reseaux-de-neurones-convolutifs-cnn-le-guide-ultime-pour-une-croissance-rapide-dans-le-deep-learning/ [En ligne, consulté le 10/04/2025]

[W9]: Allez plus loin en Deep Learning avec les réseaux RNN, https://france.devoteam.com/paroles-dexperts/aller-plus-loin-en-deep-learning-avec-les-reseaux-de-neurones-recurrents-rnns/ [En ligne, consulté le 10/04/2025]

[W10] : Deep Learning architectures, https://developer.ibm.com/articles/cc-machine-learning-deep-learning-architectures/ [En ligne, consulté le 01/04/2025]

[W11] : Architecture de GRU, https://developer.ibm.com/developer/default/articles/cc-machine-learning-deep-learning-architectures/images/figure05.png [En ligne, consulté le 20/04/2025]

[W12] : Architecture de MLP, https://www.datacamp.com/fr/tutorial/multilayer-perceptrons-in-machine-learning [En ligne, consulté le 20/04/2025]

[W13] : Les frameworks les plus utilisés en Deep Learning https://devopedia.org/deep-learning-frameworks, [En ligne, consulté le 04/10/2025]

[W14] : Le dataset ember https://github.com/elastic/ember?tab=readme-ov-file [En ligne, consulté le 10/08/2025]

[W15]: Dataset CIC-Malmem-2022 https://www.unb.ca/cic/datasets/malmem-2022.html , [En ligne, consulté le 20/10/2025]

[W16]: Détail sur les matrices de confusion, https://www.picsellia.fr/post/matrice-confusion-computer-vision , [En ligne, consulté le 05/11/2025]

MERCI POUR VOTRE

AIMABLE ATTENTION !!!