h e g

Haute école de gestion
Genève

L'utilisation des réseaux neuronaux pour la détection d'intrusions dans les réseaux informatiques

Travail de Bachelor réalisé en vue de l'obtention du Bachelor HES

par :

Maxime Perrod

Conseiller au travail de Bachelor :

Jean-Luc Sarrade, chargé de cours HES

Genève, le 22 août 2023

Haute École de Gestion de Genève (HEG-GE)

Filière Informatique de gestion

Hes·so GENÈVE
Haute Ecole Spécialisée
de Suisse occidentale

Déclaration

Ce travail de Bachelor a été réalisé dans le cadre de l'examen final de la Haute école de gestion de Genève, en vue de l'obtention du titre « Bachelor of Science en Informatique de Gestion. »

L'étudiant a envoyé ce document par courriel à l'adresse remise par son conseiller au travail de Bachelor pour analyse par le logiciel de détection de plagiat COMPILATIO, selon la procédure détaillée sur Cyberlearn.

L'étudiant accepte, le cas échéant, la clause de confidentialité. L'utilisation des conclusions et recommandations formulées dans le travail de Bachelor, sans préjuger de leur valeur, n'engage ni la responsabilité de l'auteur, ni celle du conseiller au travail de Bachelor, du juré et de la HEG.

« J'atteste avoir réalisé seul le présent travail, sans avoir utilisé d'autres sources que celles citées dans la bibliographie. »

Fait à Genève, le 22 août 2023

Perrod Maxime

Remerciements

Je tiens tout d'abord à remercier Jean-Luc Sarrade, mon professeur accompagnant pour la réalisation de ce travail. Il a su être présent tout au long, répondant à mes questions et me fournissant des conseils très pertinents tout au long du projet, ainsi que pendant les périodes de vacances. Merci, M. Sarrade, d’avoir fait de cette expérience académique une aventure à la fois enrichissante et inspirante.

Je tiens également à exprimer ma gratitude envers mon père et M. Yves Balet pour leurs précieuses relectures et leurs observations judicieuses lors de la finalisation de ce travail académique.

Résumé

Dans la première partie consacrée à l'état de l'art, le document est divisé en deux sections principales : la cybersécurité et le machine learning.

- **Cybersécurité** : Cette section analyse neuf catégories différentes d'attaques informatiques. Elle explique non seulement les concepts, mais aussi le fonctionnement de chaque type d'attaque. Par la suite, nous discuterons des différents systèmes existants pour la détection des intrusions, offrant un aperçu complet du paysage de la cybersécurité.
- **Machine Learning** : La deuxième section de l'état de l'art se concentre sur le Machine Learning, en commençant par une introduction générale à ce domaine. Nous plongerons ensuite dans le fonctionnement du Deep learning, avec une attention particulière portée aux réseaux de neurones convolutifs (CNN). Le fonctionnement, la structure, et l'application des CNN sont expliqués en détail, fournissant une base pour la suite du travail.

Après avoir établi ses fondations théoriques, le travail pose une problématique spécifique. La réponse à cette question est explorée dans la méthodologie et la réalisation d'un travail pratique. Cette partie met en évidence la conception, le développement, et l'évaluation d'un algorithme de machine Learning adapté à la tâche de détection des menaces informatique.

En conclusion, ce travail de bachelor représente une étude complète et détaillée de l'intersection entre la cybersécurité et le machine Learning. Il contribue à la fois à la compréhension théorique et à l'application pratique de ces concepts.

Table des matières

Liste des figures

1 Introduction

1.1 Contexte et motivation

La cybersécurité est devenue un enjeu crucial dans le contexte actuel de notre société fortement interconnectée. Une variété d'entreprises, indépendamment de leur taille ou de leur secteur d'activité, se trouve confrontée aux risques liés aux attaques numériques. Les conséquences de ces intrusions peuvent être extrêmement préjudiciables, allant du vol de données ou de propriété intellectuelle à des perturbations importantes de leurs services.

Dans ce contexte, l'intégration de systèmes de détection d'intrusion (IDS) dans les politiques de sécurité des entreprises s'avèrent essentielles. Les systèmes privilégiant les approches basées sur les signatures et les heuristiques dominent le paysage actuel. Cependant, des solutions qui exploitent le machine learning sont à l'étude et pourraient représenter l'avenir de la cybersécurité.

L'intelligence artificielle, et plus précisément les branches du machine learning et du deep learning, suscite un intérêt considérable dans divers domaines, y compris la cybersécurité. Ces technologies dotées de la capacité d'adaptation, ce qui permettrait aux systèmes d'évoluer et de s'adapter aux nouvelles menaces.

En tant qu'étudiant intéressé par l'IA et la cybersécurité, ce travail de bachelor représente une opportunité unique de se spécialiser dans ces domaines et d'acquérir des compétences précieuses pour faire face aux défis du monde numérique d'aujourd'hui et de demain.

1.2 Objectifs du travail de bachelor

Le principal objectif de ce travail de bachelor est d'étudier l'application du machine learning et plus particulièrement les réseaux neuronaux pour la détection d'intrusions dans les réseaux informatiques. L'intention est de développer un modèle de machine learning permettant d'identifier les infractions dans les systèmes informatiques.

État de l'art et Fondements théoriques

1.1 La cyber sécurité

La cybersécurité est un ensemble de pratique, et de technologie visant à protéger les ordinateurs, les serveurs, les appareils mobiles, les systèmes électroniques, les réseaux et les données contre les attaques malveillantes [1]. La cybersécurité est un domaine qui comporte plusieurs spécialisations diverses, telles que :

- Sécurité des réseaux : Protège l'infrastructure réseau contre les menaces et les vulnérabilités, en assurant l'intégrité et la confidentialité des données transmises.
- Sécurité des applications : Se concentrent sur la sécurisation des applications et des logiciels contre les exploitations, les injections de code malveillant et autres attaques.
- Sécurité de l'information : Vise à protéger les informations sensibles des organisations, en mettant en place des politiques, des procédures et des contrôles techniques. [2]

La cybersécurité a pris naissance en 1970 avec la création d'un programme appelé « Creeper » par le chercheur Bob Thomas. Ce programme était conçu pour se propager d'un ordinateur à un autre sur le réseau ARPANET, considéré comme l'ancêtre d'Internet. Creeper affichait un message chaque fois qu'il infectait un nouveau système, disant : "I'm the creeper, catch me if you can !" (Je suis le Creeper, attrape-moi si tu peux !). Bien qu'il n'ait pas été conçus pour causer des dommages. Le chercheur Ray Tomlinson a développé le premier programme antivirus, nommé « Reaper ». Reaper, tout comme Creeper, pouvait se déplacer sur le réseau ARPANET, mais il avait pour mission de détecter et d'éliminer Creeper des systèmes infectés [3, 4].

Cet épisode a marqué le commencement de la rivalité entre les attaquants et les défenseurs dans le domaine de la cybersécurité. Au fil des ans, la complexité et la sophistication des attaques informatiques ont considérablement augmenté, entraînant une prise de conscience croissante de la nécessité de protéger les systèmes informatiques et les données.

Aujourd'hui, la cybersécurité est un enjeu mondial, la numérisation rapide de notre société rendant nos systèmes informatiques et nos infrastructures critiques et vulnérables à une multitude de cyberattaques. Les entreprises et les gouvernements doivent désormais investir dans la cybersécurité afin de se protéger contre les menaces en constante évolution pesant sur notre monde connecté.

Les experts en cybersécurité font tout pour essayer de combler les lacunes, mais les pirates informatiques sont constamment à la recherche de nouvelles méthodes afin de contourner les dispositifs de défense et d'exploiter toutes les nouvelles failles de sécurité qui se présentent. Les dernières vulnérabilités nécessitent une réflexion plus poussée, car elles tirent parti des nouvelles conditions de travail à domicile, des outils de télétravail et des services infonuagiques [2]. Ces menaces en constante évolution incluent :

- Logiciels malveillants
- Ransomware
- Hameçonnage/ingénierie sociale
- Menaces internes
- Attaques par déni de service distribué (DDoS)
- Menaces persistantes évoluées
- Attaques de l'homme du milieu

Les cyberattaques visent à obtenir accès à des données, à les modifier ou les détruire, à réclamer des fonds aux utilisateurs ou à l'organisation, ou à perturber le bon fonctionnement de l'entreprise. [1]

« En 2020, le coût moyen d'une atteinte à la protection des données était de 3,86 millions de dollars dans le monde, et de 8,64 millions de dollars aux États-Unis. » [1]

Ces dépenses comprennent les coûts associés à la détection et à la gestion des failles de sécurité, les pertes en termes d'investissements et de revenus, ainsi que l'impact négatif durable sur la réputation et l'image de marque d'une entreprise. C'est pourquoi il est essentiel de se focaliser sur la cybersécurité [1].

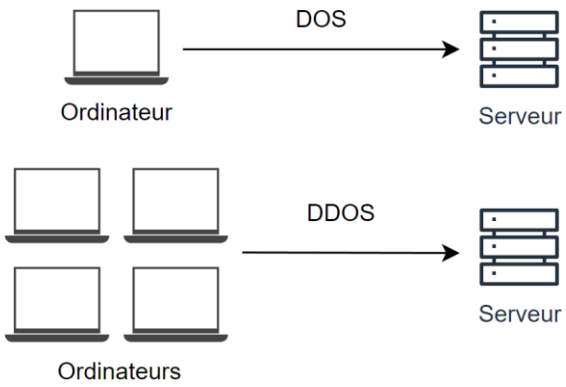
1.1.1 Menaces et attaques courantes

Dans le cadre de ce mémoire, nous nous concentrerons sur les menaces et les attaques les plus fréquentes en matière de cybersécurité dans le domaine des réseaux.

1.1.1.1 Attaques par déni de service (DoS) :

L'objectif des attaques par déni de service est de rendre un service indisponible, empêchant ainsi les utilisateurs d'y accéder. Les attaques DoS fonctionnent généralement en submergeant un service de requêtes jusqu'à saturation, de sorte que le trafic normal ne peut plus être traité. Un DoS implique l'utilisation d'un seul ordinateur. Aujourd'hui, la majorité de ces attaques sont réalisées de manière distribuée, d'où l'appellation « DDoS » pour Distributed Denial of Service Attack. [5]

Figure 1 - DOS et DDOS



(Inspiré de [8])

Attaques par déni de service distribué (DDoS):
Les attaques DDoS sont exécutées à l'aide de réseaux d'ordinateurs et d'autres équipements IoT. Ces machines sont souvent infectées par un logiciel malveillant qui crée un groupe de bots « zombies », appelé botnet. Une fois le botnet constitué, le pirate envoie des instructions d'attaque au botnet. Comme chaque bot possède une adresse IP différente, il est difficile pour le serveur de distinguer les attaques du trafic normal [6].

Smurfing
Le smurfing est une forme spécifique d'attaque DDoS qui vise à saturer un serveur en utilisant des requêtes ICMP (Internet Control Message Protocol). Dans cette attaque, l'assaillant utilise une fausse adresse IP d'expéditeur, correspondant à l'adresse IP de la victime. Il envoie ensuite des requêtes ICMP à tous les hôtes du réseau. Les hôtes du réseau, trompés par la fausse adresse IP d'expéditeur, sont amenés à répondre simultanément à ces requêtes en renvoyant des réponses à l'adresse IP de la victime. Cela génère un effet d'amplification qui peut saturer les ressources réseau de la victime, entraînant potentiellement une indisponibilité de ses services [7].

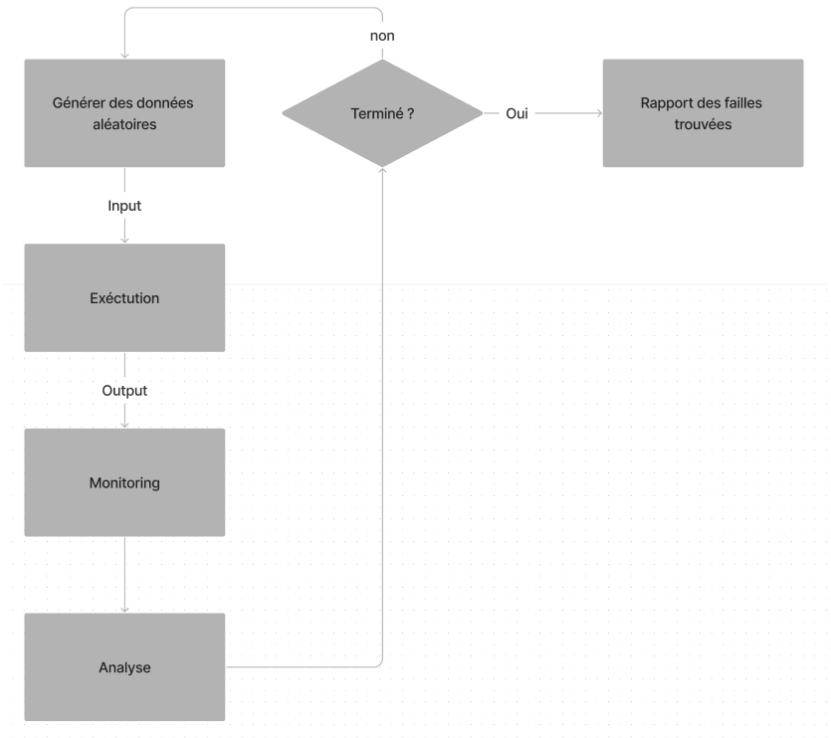
Le principal symptôme d'une attaque par déni de service (DoS) est la lenteur ou l'indisponibilité subite d'un service. Cependant, il est important de noter qu'un pic de trafic légitime peut également causer ces problèmes, d'où la nécessité d'approfondir les investigations.

- Des volumes suspects de trafic provenant d'une seule adresse IP ou d'une plage d'adresses IP.
- Des volumes suspects provenant d'utilisateurs ayant le même profil comportemental, comme le même type d'appareil, la même géolocalisation ou le même navigateur web.
- Une augmentation des requêtes sur une seule page web ou un seul point de terminaison.
- Des modèles de trafic inhabituels, tels que des pics à des heures inhabituelles ou des schémas récurrents, comme des pics toutes les 10 minutes [6].

1.1.1.2 Le fuzzing :

Le fuzzing est une méthode permettant de détecter les failles d'un système informatique en insérant des données semi-aléatoires dans diverses entrées d'un programme. Une fois les données insérées, l'état du programme est surveillé : Exceptions, plantages, fuite de mémoire et des comportements inattendus. Les failles sont les principales sources de vulnérabilité d'un système et ces attaques sont particulièrement redoutées, car elles peuvent révéler des vulnérabilités inconnues, offrant ainsi aux attaquants une porte d'entrée potentielle [9, 12].

Figure 2 - Diagramme du fonctionnement du Fuzzing



(Inspiré de [12])

Le fuzzing peut être utilisé dans divers cas, tels que :

- **Application :**
  - Tester toutes les combinaisons de boutons/entrées de texte
  - L'import/ export de fichier en essayait différent type de fichier ou des fichiers modifier

  Pour les applications web : les URL, formulaires, contenu généré par l'utilisateur, les requêtes.
- **Protocol Fuzzing :**
  C'est celui qui nous intéresse le plus lors de ce travail. Il existe beaucoup de protocole. Le « protocol fuzzing » consiste à tester le comportement d'un serveur avec un protocole donné, tel que HTTP. La tâche du chercheur de faille est de trouver les vulnérabilités dans les implémentations du protocole. Le pirate ajouter une valeur aléatoire à chaque valeur d'en-tête HTTP ce qui permettra de découvrir des failles [10, 11].

1.1.1.3 Ports scan :

Le scan des ports est une technique qui consiste à envoyer des paquets de données vers divers ports d'un système, dans le but de déterminer quels services sont actifs et si les ports sont ouverts ou fermés. Cette technique est utilisée par les administrateurs réseau pour vérifier la sécurité de leurs réseaux.

Cependant, le balayage de ports est également un outil fréquemment utilisé par les pirates. Ils s'en servent pour identifier les vulnérabilités potentielles dans un système, en se basant sur les informations recueillies durant le balayage. Ces données permettent ensuite aux pirates de planifier leurs attaques de manière plus stratégique et ciblée [14, 15].

Lors d'une analyse de port, l'utilisateur envoie des requêtes avec divers protocoles comme TCP ou UDP. En fonction de ce que l'utilisateur cherche à découvrir. Si le port est ouvert, il renverra une réponse. Dans le cas contraire, le port est fermé. L'analyse prend note des ports qui répondent et semblent vulnérables [13].

L'analyse classera les ports en 3 catégories :

- Ouvert :
  Le port est ouvert et signifie qu'un service est en train d'écouter sur le port et il est prêt à accepter des connexions. C'est potentiellement un point d'entrée.
- Fermé :
  Le port est fermé et signifie qu'il n'y a pas d'application ou de service en cours d'écoute sur ce port à ce moment précis.
- Filtré :
  L'hôte ne répond pas à la demande. Cela peut signifier que le paquet a été abandonné en raison de problème réseau ou d'un pare-feu. [15]

Pour se prémunir contre les balayages de ports, il est crucial d'instaurer des mesures de sécurité appropriées, telles que l'utilisation de pare-feu. Un pare-feu joue un rôle essentiel en bloquant l'accès non autorisé à un réseau privé. Il peut également détecter un balayage de ports en cours et l'interrompre, renforçant ainsi la sécurité du réseau.

Une autre stratégie consisterait à mettre en place des "TCP wrappers". Ceux-ci offrent aux administrateurs la possibilité de contrôler l'accès aux serveurs en fonction de l'adresse IP de l'interlocuteur. Ils peuvent ainsi autoriser ou refuser l'accès, fournissant une couche supplémentaire de sécurité.

De plus, il est important de réaliser régulièrement des balayages de ports internes. Cette pratique aide à identifier les éventuelles faiblesses du réseau qui pourraient être exploitées par des pirates [15].

Les variétés de scans de ports sont détaillées en annexe :
Annexe 2 : Description détaillée des colonnes du jeu de données.

1.1.1.4 Backdoors (Portes dérobées) :

Les attaques par backdoor sont des méthodes d'infiltration qui exploitent des failles spécifiques des logiciels, permettant ainsi aux cybercriminels de pénétrer un système en évitant les mécanismes de sécurité standard. Une fois qu'une backdoor est découverte, les pirates tendent à la garder secrète pour continuer à en bénéficier [17].

Ces portes dérobées sont généralement mises en place par des logiciels malveillants, notamment les Remote Access Trojans (RAT). Conçus pour infiltrer et prendre le contrôle d'un système ou d'un réseau, les RAT peuvent s'immiscer au cœur d'un système, installer une backdoor, et ainsi donner à un cybercriminel la possibilité de manipuler le système à distance. Ils se propagent souvent via des logiciels malveillants qui se font passer pour des programmes légitimes [16, 17].

Les portes dérobées sont dangereuses, car une fois mises en place. Elles peuvent :

- Diffuser des programmes malveillants
- Faciliter le moyen de mener une attaque DDoS
- Modifier des paramètres sensibles tel que les mots de passe.

Les attaquants peuvent installer, exécuter des applications spécifiques à l'aide de backdoors.

Les backdoors représentent donc un risque majeur pour la sécurité informatique. Il est important de prendre des mesures afin de s'en prévenir. Ces mesures peuvent inclure le changement des mots de passe par défaut, la surveillance constante du réseau informatique, l'activation permanente du pare-feu et une vigilance accrue quant aux applications installées sur les systèmes [17].

1.1.1.5 Exploits :

Exploit dérive du verbe "exploiter", qui implique l'utilisation d'une chose à son profit. Un exploit est une forme de code spécifiquement conçu pour tirer avantage d'une vulnérabilité ou d'une faille de sécurité afin de provoquer des comportements inattendus dans un système. Ces exploits peuvent être créés par des chercheurs en sécurité ou par des individus mal intentionnés [18].

L'utilisation d'un exploit permet à un individu malveillant d'accéder à distance à un réseau, d'obtenir des privilèges élevés ou de pénétrer plus profondément dans le système. De plus, il est envisageable de combiner plusieurs exploits pour d'abord accéder au réseau à un niveau inférieur, puis progressivement élever les privilèges jusqu'à atteindre le niveau le plus élevé, notamment en accédant aux privilèges administratifs [19].

Lorsqu'un exploit est dévoilé, la faille de sécurité est souvent corrigée par une mise à jour du logiciel, ce qui rend l'exploit inefficace sur les versions mises à jour du logiciel. C'est pour cette raison que les cybercriminels choisissent souvent de ne pas divulguer leurs exploits. Ces vulnérabilités non révélées sont appelées failles "zero-day" [19].

1.1.1.6 Reconnaissance :

C’est une technique permettant de collecter des informations sur un système cible. Il existe différents types d’attaque de reconnaissance :

1. Enumération DNS est une méthode de reconnaissance qui consiste à interroger un serveur associé à un site internet ou à un hébergeur. Les DNS offrent la possibilité de recueillir des informations spécifiques telles que les adresses IP, les noms des serveurs et les fonctions de ces derniers, par exemple déterminer quel serveur est en charge des e-mails.

   Il existe de nombreuses manières et commandes pour obtenir des informations en communiquant avec le serveur DNS.

   Une des techniques couramment utilisées est le transfert de zone DNS. Cette opération réalise une copie intégrale des données DNS. Si un serveur DNS est mal configuré et autorise les copies avec un serveur non approuvé, alors un assaillant peut avoir accès à une multitude d'informations : les correspondances entre les noms d'hôtes et leurs adresses IP, le serveur de courrier, le serveur DNS du domaine, ainsi que des informations administratives sur le domaine, y compris l'adresse e-mail de l'administrateur du domaine. [20]

2. L'énumération SMTP est une technique permettant d'identifier les utilisateurs valides sur un serveur SMTP. Cette technique peut être utilisée pour préparer des attaques de phishing. Le SMTP (Simple Mail Transfer Protocol) est un protocole de communication utilisé pour les e-mails. [21]

   Il existe plusieurs techniques d'énumération SMTP :
   - La commande VRFY permet de vérifier si un utilisateur spécifique existe.
   - La commande EXPN permet de récupérer toutes les adresses e-mail liées à une liste de diffusion.

3. Sun RPC, ou Sun Remote Procedure Call, est une technologie qui permet à un programme informatique d’appeler une fonction d'un autre programme se trouvant sur un autre ordinateur connecté au réseau.

   Le portmapper est un service qui facilite la mise en œuvre du protocole RPC. Les programmes souhaitant proposer un service RPC s'inscrivent auprès du portmapper en précisant le numéro de port sur lequel ils sont en attente d'une requête. Les programmes clients peuvent alors consulter le portmapper pour découvrir le port auquel ils doivent se connecter pour accéder à un service RPC spécifique.

   Un attaquant pourrait interroger le portmapper et obtenir une liste des services RPC disponibles, fournissant à l'assaillant des informations sur les services susceptibles d'être vulnérables. [22]

1.1.1.7 Shellcode :

Le shellcode, dans le contexte du piratage, est un fragment de code utilisé comme payload lors de l'exploitation d'une faille logicielle. Il porte le nom de "shellcode", car il lance généralement un shell de commandes (terminal) à partir duquel l'attaquant peut prendre le contrôle de la machine compromise. [23]

L'élaboration d'un shellcode requiert une connaissance approfondie du langage assembleur propre à l'architecture cible. Habituellement, un shellcode spécifique est nécessaire pour chaque version de chaque système d'exploitation et pour chaque type d'architecture matérielle. Des appels système sont utilisés dans le shellcode pour accomplir des actions. Par conséquent, la majorité des shellcodes sont dépendants du système d'exploitation, étant donné que ces appels système diffèrent d'un système à l'autre. [23]

Le shellcode est exécuté lorsqu'une vulnérabilité a été exploitée. Généralement, le shellcode est limité par des contraintes de taille, comme la taille d'un tampon envoyé à une application vulnérable. Plusieurs stratégies peuvent être adoptées dans la création d'un shellcode. Parmi celles-ci, l'efficacité, qui consiste à réduire la taille du shellcode ; la polyvalence, qui permet d'utiliser un proxy d'appel système pour augmenter la flexibilité tout en contournant certaines mesures de sécurité ; l'obfuscation, qui rend le shellcode polymorphe pour le protéger contre la détection par les antivirus ; et enfin, le tunnel crypté, qui permet de créer une connexion cryptée entre l'attaquant et la machine cible, rendant les données échangées difficilement lisibles. Ces stratégies peuvent être combinées en fonction des besoins. [23]

Un shellcode ou une exploitation doit être fiable, car dans le cas contraire, il pourrait entraîner une défaillance de l'application ciblée. Dans une telle situation, l'administrateur du système se demanderait sans doute pourquoi celui-ci a planté et chercherait à identifier le problème. De plus, un shellcode non fiable pourrait aussi corrompre la mémoire de l'application. Pour pouvoir réutiliser l'exploit, un redémarrage du système serait alors nécessaire. Cependant, dans un environnement de production, un redémarrage pourrait n'être programmé que plusieurs mois plus tard. Pour ces raisons, il est essentiel qu'un shellcode soit précis et fiable. [24]

Il y a plusieurs types de shellcode. Il peut être local ou distant. [23]

- Locale :
  Le shellcode local est utilisé lorsque l'attaquant a un accès limité à une machine, mais peut exploiter une vulnérabilité, par exemple un débordement de tampon, pour obtenir des privilèges plus élevés.
- À distance :
  Le shellcode distant, quant à lui, est utilisé lorsqu'un attaquant souhaite cibler un processus vulnérable sur une autre machine via un réseau

1.1.1.8 Worms :

L'expression "worms", qui se traduit par "ver" en français, se réfère à un type particulier de logiciel malveillant. À l'image d'un virus biologique, un virus informatique ne peut pas subsister s'il infecte une entité non vivante. Son activation ne peut se faire qu'à travers l'exécution d'un programme. Par exemple, sur Linux, des commandes comme 'ls' ou 'dir' peuvent déclencher le lancement du programme infecté. [25]

Il est important de souligner qu'aujourd'hui, les virus informatiques traditionnels sont presque obsolètes. À la différence des virus, les vers n'ont pas besoin d'un programme hôte pour fonctionner. Ils sont autonomes, ce qui leur permet de conduire activement des attaques sans dépendre d'une action extérieure. [26]

Selon les spécialistes de la cybersécurité, les attaques par vers constituent le plus grand risque pour la sécurité des réseaux informatiques. Les vers sont conçus pour se propager sans nécessiter d'interaction de la part de l'utilisateur [27]. Ils exploitent souvent les réseaux informatiques, les failles logicielles, les courriels, les dossiers partagés, mais aussi les pages web. Certains vers sont combinés avec des scripts de pages web et sont dissimulés dans des pages HTML parmi d'autres technologies. Lorsqu'un utilisateur visite une page web infectée par un tel ver, celui-ci s'installe automatiquement dans la mémoire de l'ordinateur.

Les vers ont la capacité de modifier ou de supprimer des fichiers, voire d'injecter d'autres logiciels malveillants dans un ordinateur. L'objectif d'un ver peut consister à se dupliquer lui-même, mais aussi à épuiser les ressources d'un ordinateur, en surchargeant l'espace disque, la bande passante ou le réseau partagés. De plus, ils peuvent être utilisés pour dérober des données ou installer une porte dérobée sur le système. Dans la plupart des cas, les vers provoquent des dommages au réseau en consommant une grande partie de la bande passante. [26, 28]

Un exemple marquant est celui du ver 'Slammer' qui a exploité une faille dans le logiciel de base de données SQL de Microsoft. Cela a déclenché une suite d'effets en chaîne non anticipés sur les infrastructures électroniques. Les systèmes de réservation de billets d'avion et les guichets automatiques de banque ont été parmi les plus affectés. De manière significative, le ver 'Slammer' a également perturbé les systèmes de contrôle de la centrale nucléaire de Davis-Besse dans l'Ohio. Les réseaux dits 'critiques' sont généralement les cibles privilégiées de ces attaques. Les réseaux critiques sont ceux dont une interruption prolongée (de plusieurs jours, semaines, ou indéfiniment) ou un dysfonctionnement pourrait gravement perturber la vie quotidienne. [27]

1.1.1.9 Générique :

Ce sont les attaques que l'on ne peut pas classer dans les autres catégories que l'on a précédemment vues.

- L'injection SQL est une menace importante pour la sécurité des sites web. Elle donne aux cybercriminels la possibilité d'interagir directement avec la base de données d'une application. Lorsqu'elle est exploitée, cette vulnérabilité peut permettre à un individu mal intentionné non seulement de consulter des informations auxquelles il ne devrait normalement pas avoir accès, mais aussi de les altérer ou de les supprimer.

  Le danger d'une telle attaque est considérable. En effet, par le biais d'une injection SQL réussie, des données sensibles telles que des mots de passe, des numéros de carte de crédit ou d'autres informations personnelles peuvent tomber entre de mauvaises mains.

  Bien que plusieurs outils, comme Burp Suite, soient disponibles pour détecter automatiquement ces vulnérabilités, une approche manuelle peut également s'avérer efficace. Un utilisateur souhaitant tester la résistance d'une application face à ce type d'attaque peut essayer d'insérer des caractères tels que « ' » pour observer les réactions du système. Des inputs comme « OR 1=1 » ou « OR 2=2 » peuvent également être utilisés pour identifier des comportements anormaux en comparant les résultats obtenus. L'utilisation de clauses comme "order by" peut aussi révéler des informations sur la structure de la base de données.

  La meilleure défense contre les injections SQL est d'adopter une approche préventive, en privilégiant l'utilisation de requêtes paramétrées, évitant ainsi la simple concaténation de chaînes dans les requêtes. [29]

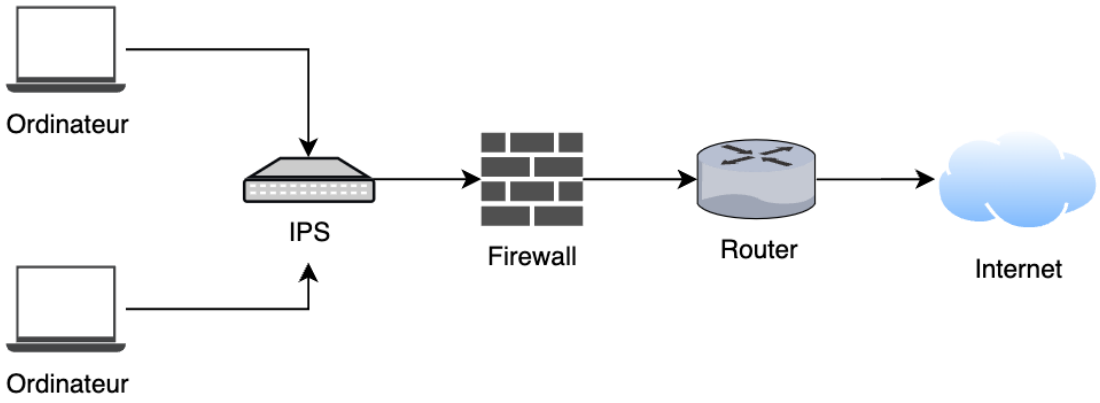
1.2 Méthodes traditionnelles de détection d'intrusion

Plusieurs dispositifs sont disponibles pour assurer la sécurité d'un réseau, parmi lesquels figurent l'IDS (Système de Détection d'Intrusion), l'IPS (Système de Prévention des Intrusions) et les pare-feu. Un pare-feu, en particulier, contrôle et filtre le trafic réseau. Il régule ce trafic en fonction de règles préalablement établies. Il se base sur les adresses IP, qu'elles soient de sources ou de destination, ainsi que sur les ports. Ainsi, tout trafic ne correspondant pas à ces critères définis peut être systématiquement rejeté. [30]

Les IDS sont des dispositifs de surveillance réseau. Leur mission principale est d'examiner le trafic réseau pour déceler les menaces ou les infractions aux règles établies. Dès qu'un élément suspect est repéré, des alertes sont émises sans délai. En règle générale, chaque activité est consignée, permettant ainsi à l'administrateur d'effectuer ultérieurement une enquête approfondie sur ces alertes et d'adopter les mesures correctives nécessaires. [31]

Les IPS sont des dispositifs complémentaires aux pare-feu. Ils surveillent les paquets du réseau en temps réel, s'il détecte quelque chose, il bloque automatiquement l'attaque. [30]. L'image ci-dessous montre comment l'IPS est installé sur le réseau.

Figure 3 - Schéma de l'IPS

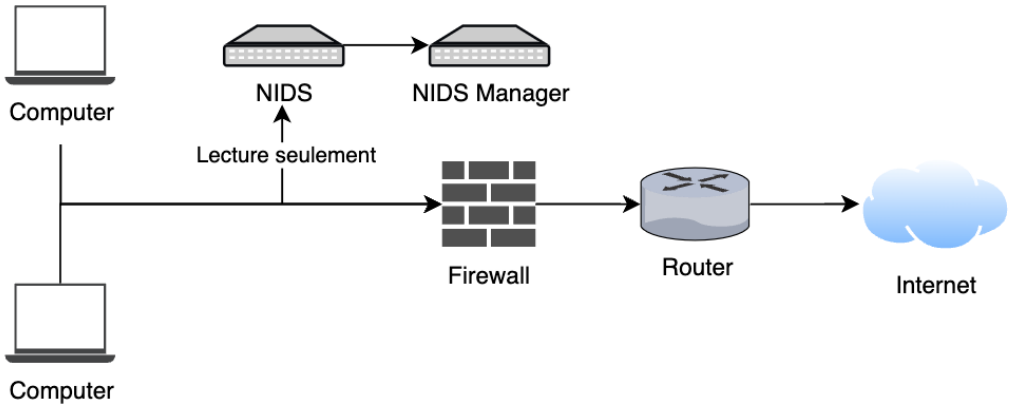


(Inspiré de [30])

Les méthodes de détection d'intrusion (IDS) se classifient généralement en deux catégories principales, adaptées à différentes échelles et contextes. Mais nous pouvons également les classer par approche de détection que l'on verra par la suite.

- **Systèmes de détection d'intrusion réseau (NIDS) :**
  Ces systèmes sont positionnés dans un sous-réseau spécifique, leur permettant d'analyser et de surveiller le flux de données. Dès qu'un risque ou une menace potentielle est repéré, une notification est transmise à l'administrateur réseau. [31]

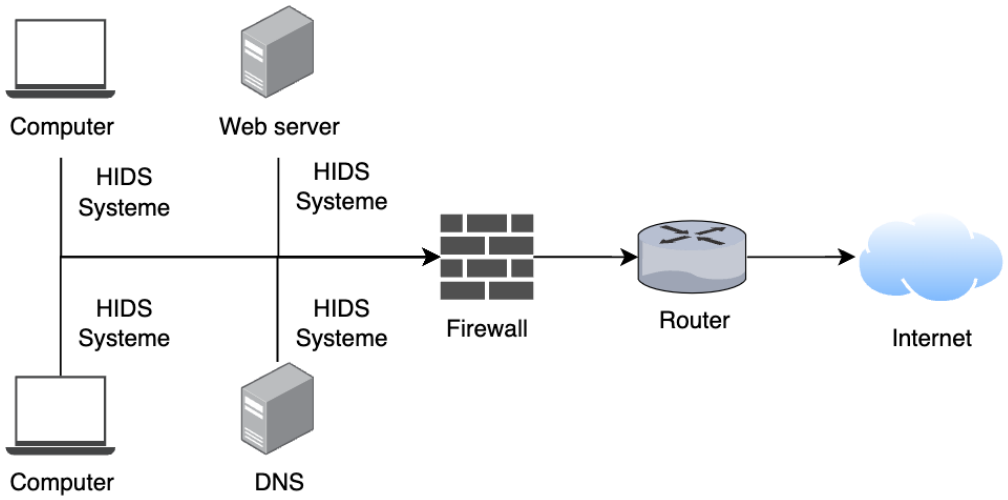
Figure 4 - Schéma de NIDS



(Inspiré de [31])

- **Système de détection d'intrusion basé sur l'hôte (HIDS) :**
  Ce type de système est installé sur des dispositifs spécifiques au sein du réseau. Un HIDS contrôle le trafic entrant et sortant d'un seul appareil. Lorsque des comportements anormaux sont repérés, une alerte est envoyée à l'administrateur. De plus, lors de la détection d'une activité suspecte, le HIDS fait une comparaison entre l'état actuel des fichiers système et leur état précédent. Si des modifications ou suppressions sont constatées, une alerte est déclenchée pour inciter l'administrateur à approfondir son investigation. [31]

Figure 5 - Schéma HIDS



(Inspiré de [31])

1.2.1 Approche basée sur la signature

Originellement, les concepteurs de logiciels antivirus ont mis en place le système de signature d'attaque pour examiner les fichiers système en quête de menace. Un système de détection d'intrusion (IDS, IPS) axé sur la signature se focalise habituellement sur le trafic réseau entrant, à la recherche de modèles ou de séquences d'octets qui correspondent à une signature d'attaque spécifique. Ces éléments peuvent être repérés dans les en-têtes de paquets de réseau, ainsi que dans les suites de données correspondant à des logiciels malveillants reconnus ou d'autres types de schémas malveillants. Ils peuvent également être détectés dans les adresses de réseau source ou de destination, ou dans des suites spécifiques de données.

Les IDS qui se fondent sur les signatures utilisent des bibliothèques ou des listes de signatures connues pour identifier les menaces. Ces signatures peuvent inclure un large éventail d'indicateurs de compromission (IoC) qui peuvent comprendre des suites spécifiques d'octets dans les fichiers ou le trafic réseau, des comportements spécifiques d'attaque de réseau, des domaines, des URL, des objets d'e-mails spécifiques, des hachages de fichiers. Lorsqu'une menace est identifiée, une signature est générée et ajoutée à la liste. Les solutions utilisant ce système ont des taux de détection de menace élevés sans faux positifs, car toutes les alertes sont générées sur la base de contenu malveillant connu. [32]

Néanmoins, la principale limite des solutions fondées sur la signature est leur incapacité à repérer les attaques inconnues. En effet, les cybercriminels peuvent aisément modifier leurs séquences d'attaque pour éviter d'être détectés. [33]

1.2.2 Approche par anomalie

L'IDS/ IPS basé sur les anomalies a été développé avec l'objectif de repérer les attaques encore non identifiées. Pour ce faire, il recourt à l'apprentissage automatique pour établir un modèle de comportement de confiance, contre lequel toutes les activités sont ensuite comparées. Toute activité ne correspondant pas à ce modèle est considérée comme suspecte. [31]

Cette technique est généralement plus efficace pour déceler des attaques nouvelles, mais sa phase d'apprentissage est particulièrement consommatrice de temps. De plus, l'exactitude de la détection peut être compromise par l'occurrence de faux positifs, engendrés par un déficit d'information dans le modèle de comparaison. Des alertes erronées peuvent ainsi être envoyées à l'administrateur. Par exemple, lorsqu'un programme est mis à jour ou déployé pour la première fois, l'IDS pourrait interpréter cet événement comme anormal. [34]

1.2.3 Approche hybride

L'approche combinée envisage l'utilisation conjointe des systèmes de détection basés sur la signature et sur les anomalies. Cette méthode vise à concevoir un système de détection d'intrusion capable de repérer à la fois des attaques préalablement identifiées et celles qui demeurent encore inconnues. Elle tire profit de la précision inhérente aux signatures et de la flexibilité offerte par l'approche par anomalie.

L'outil Snort est un exemple de système de détection d'intrusion (IDS) qui intègre à la fois les capacités heuristiques et le mécanisme de signature. [34]

Snort intègre un module de détection d'anomalie (ADS), fonctionnant en deux étapes distinctes : [35]

- **Étape d'apprentissage :**
  À partir des journaux de connexion réseau, des motifs récurrents, appelés épisodes, sont extraits des données d'apprentissage. Ces motifs, représentant des comportements normaux, sont ensuite transformés en règle d'épisodes fréquents (FER).
- **Étape de détection :**
  Les FER sont élaborés en se basant sur les événements de connexion observés, puis comparés aux FER standards. Un score est calculé pour chaque FER détecté, symbolisant son niveau d'anomalie.

Ces FER contribuent par la suite à la création de nouvelles signatures et à la mise à jour régulière de la base de données de Snort, permettant ainsi une réponse plus rapide et plus précise aux menaces potentielles. [34]

De cette façon, ce système mixte permet de réduire l'intervention de l'administrateur, car la base de données est mise à jour automatiquement à la suite de chaque détection d'activité malveillante. [34]

1.3 Le Machine Learning

Le domaine de l'"apprentissage automatique", également connu sous son appellation anglaise "machine learning", repose sur des méthodes mathématiques et statistiques. Il vise à doter les ordinateurs de la capacité d'apprendre à partir des données sans être explicitement programmés. [36]

Il existe plusieurs catégories au sein du machine learning disponible en **Annexe 2 : Différents types d'apprentissage du ML**

1.3.1 Réseaux de neurones artificiels

Dans le cadre de l'apprentissage automatique, une catégorie d'algorithmes, connue sous le nom de deep learning ou apprentissage profond, est essentielle à l'élaboration d'un système intelligent. Le deep learning peut être perçu comme une boîte noire qui transforme une entrée en sortie. Ce champ d'études a suscité un vif intérêt dans le monde universitaire en raison de ses performances impressionnantes dans divers secteurs tels que la reconnaissance vocale, la vision par ordinateur et le traitement du langage naturel. [40]

En utilisant des réseaux de neurones, ces systèmes sont capables de déceler des motifs complexes au sein des données. Un avantage notable de cette technique est qu'il n'est pas indispensable de sélectionner au préalable les caractéristiques, ou "features", pertinents, comme cela serait requis avec un algorithme classique de machine learning. En effet, le deep learning a la particularité d'être capable d'identifier par lui-même les éléments les plus significatifs. [41]

1.3.1.1 Architecture

L'architecture d'un réseau de neurones s'inspire librement de la structure du cerveau biologique. Dans le cerveau humain, les neurones sont interconnectés via des synapses. De manière similaire, dans un réseau de neurones artificiels, les neurones sont liés par ce qu'on appelle des "arêtes".

Chaque neurone artificiel reçoit plusieurs signaux d'entrée, les traite et transmet son signal de sortie au neurone suivant. Ce dernier récupère une valeur numérique réelle, effectue un calcul basé sur une fonction non linéaire de la somme de ses entrées, et renvoie sa valeur via l'arête vers le prochain neurone.

Chaque arête dispose d'un "poids" associé, qui détermine le degré d'importance de ce signal. Un poids élevé signifie que le signal revêt une importance considérable pour la décision finale, tandis qu'un poids faible indique une importance moindre.

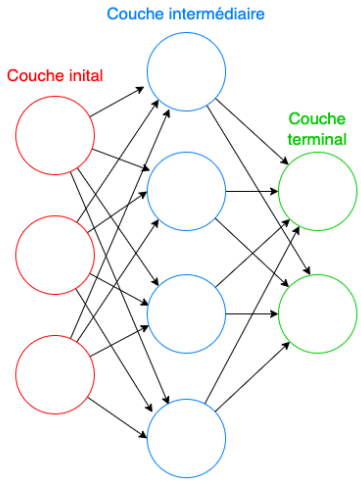
De plus, les neurones peuvent être dotés d'un seuil spécifique. Cela signifie que le signal ne sera transmis au neurone suivant que si la valeur obtenue dépasse ce seuil prédéterminé. [42]

Un réseau de neurones se compose de différentes couches :

- La couche initiale, ou couche d'entrée (input layer) est le point de départ du processus. C'est là que les informations brutes sont introduites dans le système. Par exemple, si notre objectif est de reconnaître des images de chats à travers le réseau de neurones, cette couche serait composée des pixels constituant l'image.
- S'ensuivent diverses couches intermédiaires, appelées couches cachées (hidden layers). Chacune de ces couches transforme les informations reçues de la couche précédente en fonction des paramètres spécifiques de cette couche.
- Enfin, la couche terminale, ou la couche de sortie (output layer) est chargée de la tâche finale, qui est généralement la classification. Par exemple, si l'objectif de notre réseau est de déterminer si une image est celle d'un chat ou d'un chien, cette couche pourrait comprendre deux neurones distincts : un pour "chat" et un autre pour "chien". Le neurone qui obtient le score le plus élevé déterminerait la classification finale.

De ce fait, les données injectées dans le réseau circulent de couche en couche, jusqu'à parvenir à la couche de sortie, où le résultat est généré. [41]

Figure 6 - Schéma réseau de neurones artificiels



(inspiré de [42])

L'utilisation des réseaux neuronaux pour la détection d'intrusions dans les réseaux informatiques

1.3.1.2 Formation

Les réseaux de neurones se construisent à partir d'exemples, chaque exemple comprenant une "entrée", c'est-à-dire la donnée brute, et un "résultat" attendu, soit l'issue que la donnée devrait engendrer. Ainsi, ces réseaux se forment en traitant une série d'échantillons. Par exemple, pour apprendre à identifier des chats sur des images, un réseau de neurones sera confronté à une série d'images marquées soit comme "chat" soit comme "non chat".

L'apprentissage s'opère en deux phases :

1. Le réseau saisit la donnée, la traite et tente d'en anticiper le résultat.
2. Puis, il compare sa prévision avec la valeur réelle. Si la prédiction du réseau est correcte, tout se passe bien. Sinon, il ajuste le poids de ses connexions, modifiant légèrement sa stratégie afin d'accroître ses chances de succès ultérieurs.

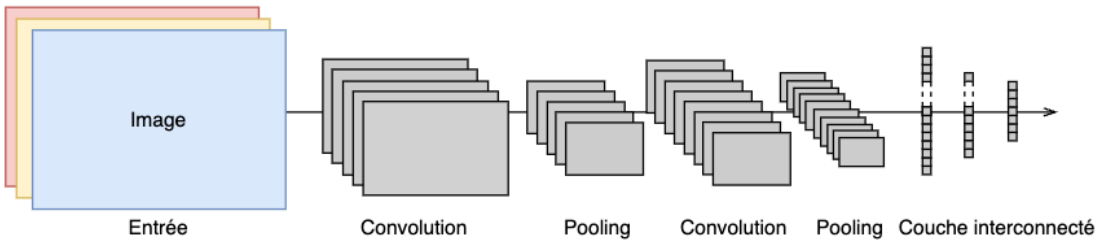
Ce processus est répété un grand nombre de fois. Au fil du temps, le réseau apprend à discerner les caractéristiques qui distinguent les différentes valeurs. À un certain stade, l'apprentissage peut être arrêté en fonction de certains critères préétablis. Ce processus constitue une forme d'apprentissage supervisé. Il est important de noter que le réseau ne possède pas au préalable de connaissance explicite sur ce qu'est un chat, comme le fait qu'il possède des moustaches ou des oreilles pointues. Il acquiert ces connaissances en examinant les exemples d'images qui lui sont présentés. [42]

1.3.2 Réseaux neuronaux convolutifs (CNN)

1.3.2.1 Introduction aux CNN

Les réseaux convolutifs, également connus sous le nom de CNN, sont une sous-catégorie de réseaux de neurones multicouches spécialement conçus pour traiter des données comme les images et les vidéos [40]. Ils sont également fréquemment utilisés en cybersécurité [43]. L'architecture des CNN est conçue pour réduire la quantité de paramètres, limitant ainsi la complexité du modèle et prévenant le surapprentissage. Cela se traduit par un modèle plus efficace en termes de calcul. Ces réseaux sont structurés en diverses couches triées selon leurs fonctionnalités, et se composent de deux blocs principaux. [40]

Figure 7 - Schéma type d'un CNN



(Inspiré de [45])

Un CNN se compose généralement de deux blocs, voici l'architecture typique d'un CNN. Elle peut décliner en diverses architectures : [44]

1. **Le bloc de convolution :**
   a. **Identification des Caractéristiques (Features)** :
      Le premier bloc est ce qui distingue le CNN des autres types de réseaux de neurones. Son rôle principal est d'identifier les caractéristiques distinctives, appelées "features". Cette identification est réalisée par une opération de convolution. Un filtre, ou un noyau, parcours l'image, effectuant un calcul pour chaque segment qu'il couvre. Cette opération est effectuée à travers plusieurs couches de convolution, chaque couche utilisant de nouveaux noyaux pour identifier des caractéristiques de plus en plus complexes.

      - **Cartes d'activation (Feature Maps) :**
        Chaque convolution crée une "carte d'activation" ou "feature map". Ces cartes indiquent où se trouvent certaines caractéristiques dans l'image. Une valeur élevée signifie que la caractéristique est présente à cet endroit.

   b. **Couche d'activation** : De plus, les couches d'activation, telles que ReLU, sont intégrées pour introduire de la non-linéarité dans le modèle. Plus précisément, la fonction d'activation ReLU élimine les valeurs négatives en les remplaçant par zéro.

c. **Couche de Pooling :**
   Entre les couches de convolution, nous trouvons fréquemment des couches de pooling, qui réduisent la dimensionnalité des données tout en préservant les informations essentielles.

2. **Le bloc entièrement connecté :**
   Le second bloc n'est pas spécifique aux CNN. Il est retrouvé à la fin de tous les réseaux de neurones utilisés pour la classification. Ici, la dernière couche, complètement connectée, transforme le vecteur d'entrée en un nouveau vecteur contenant des probabilités pour chaque classe de prédiction possible. Cette transformation emploie une fonction logistique (classification binaire) ou une fonction softmax (classification multi classes) comme fonction d'activation.

L'optimisation des paramètres dans ces réseaux est effectuée par la rétropropagation du gradient, minimisant une fonction de perte pour améliorer les performances lors de la phase d'apprentissage. [44]

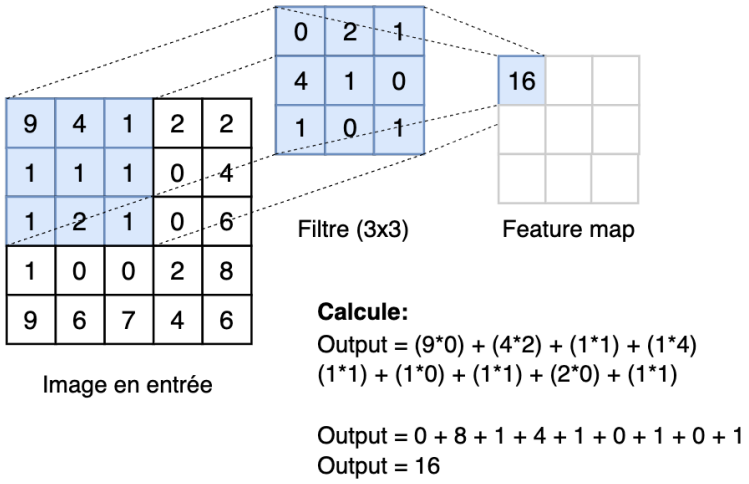
1.3.2.2 Les différentes couches d'un CNN

Les CNN, bien qu'inspirés par l'architecture des réseaux de neurones classiques, introduisent des opérations spécifiques pour traiter efficacement des données structurelles, comme les images. Trois concepts clés sont à la base des CNN : la convolution, le pooling et l'activation.

Convolution

La couche de convolution joue un rôle essentiel dans la détection des caractéristiques (ou "features") des images, telles que les contours, les formes et les textures. Les couches suivantes permettent de détecter des structures de plus en plus complexes à partir des résultats des couches précédentes. La spécificité de cette couche réside dans l'utilisation de filtres qui se déplacent sur l'image pour générer ce que l'on appelle une "feature map" ou carte de caractéristiques.

Figure 8 - Explication de la couche de convolution



(Inspiré de [46])

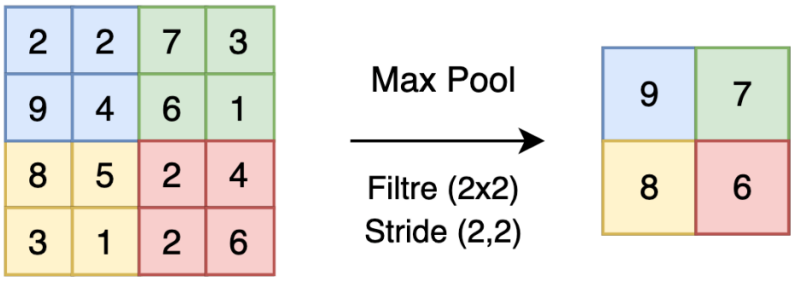
Pour illustrer le fonctionnement de la convolution, considérons une image d'entrée de dimensions 5x5. Si nous lui appliquons un filtre de taille 3x3, nous obtenons une feature map de dimensions 3x3 comme sur l'image. Le filtre exécute un produit scalaire entre ses poids et les valeurs de la portion de l'image qu'il couvre. Dans notre exemple, les valeurs du filtre (représentées en bleu) sont multipliées par les valeurs correspondantes de l'image, puis toutes ces multiplications sont sommées. Ensuite, le filtre se déplace, généralement d'un pixel à la fois, en balayant l'image de gauche à droite et de haut en bas, produisant ainsi la feature map. [46]

Couche de pooling
Intégrées entre les couches de convolution successives, les couches de pooling jouent un rôle essentiel dans les CNN. Chaque feature map produite par la couche de convolution précédente est traitée individuellement par la couche de pooling. [47]

Il y a différents types de pooling :

3. **Max Pooling** : Cette technique divise la feature map en régions distinctes et, pour chaque région, ne conserve que la valeur la plus élevée. [47]

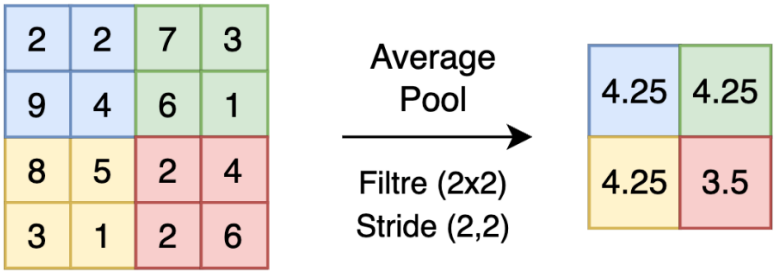
Figure 9 - Schéma du Max Pooling



(Inspiré de [47])

4. **Average Pooling** : Contrairement au max pooling, l'average pooling calcule la moyenne des valeurs dans chaque région de la feature map. [47]

Figure 10 - Schéma du Average Pooling



(Inspiré de [47])

L'objectif principal de cette opération est de réduire la dimensionnalité de l'image. Ce qui réduit le nombre de paramètres à apprendre et le nombre de calculs du réseau. [47] Malgré cette réduction, les traits ou caractéristiques les plus importantes de l'image sont conservés, assurant que l'information cruciale n'est pas perdue.

Il est important de souligner que le nombre de feature maps en sortie de la couche de pooling reste identique à celui en entrée. Cependant, leur taille est réduite en raison de l'opération de pooling.

La couche d'activation
La couche d'activation introduit une transformation non linéaire dans un réseau de neurones, modifiant la représentation spatiale des données.

Illustrons ceci avec un exemple. Prenons les chiffres 13 et 54. Dans leur représentation initiale, 13 est inférieur à 54. Si nous appliquons une fonction linéaire, comme une multiplication par 2, nous obtenons 26 et 108. Bien que leurs valeurs soient modifiées, leur relation spatiale (13 étant inférieur à 54) demeure intacte. Cependant, en utilisant une transformation non linéaire, comme le cosinus, les relations peuvent changer : cos (13) = 0.9 ce qui est supérieur cos(54) = -0.82.

Les transformations non linéaires sont importantes dans les réseaux de neurones, car elles permettent de sélectionner les informations pertinentes des données. Cela se permet d'effectuer des prédictions plus précises et robustes.

Lors de la conception d'un réseau de neurones, l'ingénieur doit choisir une fonction d'activation appropriée. Une fois sélectionnée, cette fonction est automatiquement appliquée à tous les neurones de la couche concernée. [48]

Les fonctions d'activations courantes [48] :

5. **ReLU** :
   ReLU est la fonction d'activation la plus courante en deep learning. Cette fonction élimine les valeurs négatives en les remplaçant par zéro, tout en laissant passer les valeurs positives telles quelles.

$$fonction\ ReLU(x) = max(x, 0)$$

6. **Sigmoid** :
   La fonction sigmoïde est typiquement utilisée dans la couche de sortie pour les réseaux de neurones conçus pour la classification binaire. Elle convertit les entrées en une gamme allant de 0 à 1, ce qui est interprétable comme une probabilité.

$$fonctionSigmoid(x) = 1/\big(1 + exp(-x)\big)$$

7. **Softmax** :
   La fonction softmax est privilégiée pour la couche de sortie des réseaux de neurones destinés à la classification multi classe. Elle assigne une probabilité entre 0 et 1 à chaque classe, et la somme de toutes ces probabilités est égale à 1. Cela garantit qu'une entrée est classée par rapport à toutes les classes possibles, la valeur la plus élevée indiquant la classe la plus probable.

$$fonctionSoftmax = exp(x)/sum\big(exp(x_i)\big)$$

Couche interconnectée (Dense)
La couche entièrement connectée, souvent désignée sous le nom de "couche dense", est un pilier des réseaux de neurones. Elle se distingue par le fait que chaque neurone dans cette couche est connecté à tous les neurones de la couche précédente.

Dans cet exemple, la matrice A possède des dimensions (M x N), tandis que x est un vecteur colonne de dimensions (N x 1). Les éléments de x sont des poids, des valeurs ajustables qui sont affinées pendant l'entraînement du réseau. Ce processus d'affinement s'appuie sur un algorithme appelé "rétro propagation", qui optimise ces poids pour améliorer la performance du réseau. [49]

$$Ax = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \dots & \dots & \dots & \dots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{bmatrix} \times \begin{bmatrix} x_1 \\ x_2 \\ \dots \\ x_n \end{bmatrix}$$

$$Ax = \begin{bmatrix} a_{11}x_1 & a_{12}x_2 & \dots & a_{1n}x_n \\ a_{21}x_1 & a_{22}x_2 & \dots & a_{2n}x_n \\ \dots & \dots & \dots & \dots \\ a_{m1}x_1 & a_{m2}x_2 & \dots & a_{mn}x_n \end{bmatrix}$$

L'opération clé effectuée dans la couche dense est la multiplication matricielle entre A et X. Ce produit donne en sortie un nouveau vecteur qui capture une combinaison linéaire des caractéristiques de la couche précédente, prête à être transmise à la couche suivante ou à servir de sortie du réseau. [49]

Après cette multiplication matricielle, nous avons une combinaison linéaire. Afin d'introduire une non-linéarité et d'offrir au réseau neuronal la capacité de représenter des relations plus sophistiquées, nous utilisons une fonction d'activation. La sortie de cette fonction constitue alors la sortie de la couche. [50]

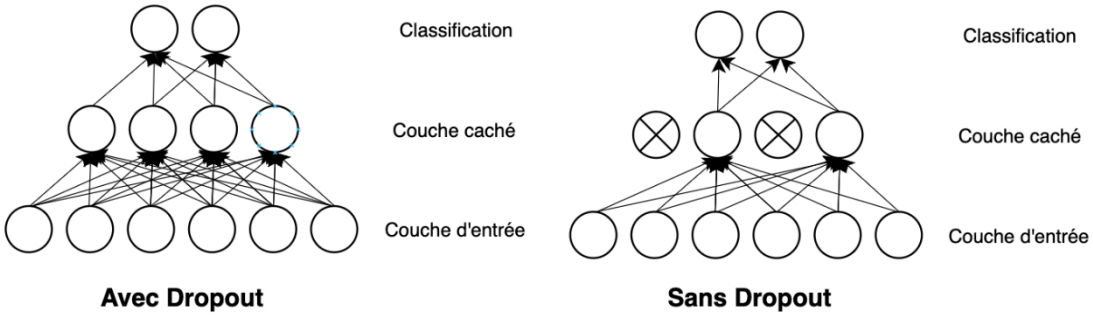
Couche de normalisation (optionnelle)
Après avoir traversé plusieurs couches, les données subissent des transformations. Ainsi, nous pourrions observer des valeurs aussi diverses que 3 ou 2867. La normalisation vise à ajuster ces valeurs à une échelle allant de 0 à 1. [50]

Couche Dropout (optionnelle)
La couche dropout est une technique qui neutralise aléatoirement la contribution de certains neurones à la couche suivante. Cette neutralisation est définie par un paramètre spécifique qui détermine le pourcentage de neurones à "éteindre". Par exemple, un taux de dropout de 0,5 signifie que 50% des neurones seront désactivés aléatoirement.

La couche dropout peut être appliquée à un vecteur d'entrée, supprimant ainsi certaines caractéristiques. Elle peut également être utilisée après une couche cachée, neutralisant dans ce cas certains neurones.

Les couches dropout sont particulièrement utiles dans les CNN (réseaux neuronaux convolutifs), car elles aident à prévenir le surapprentissage (overfitting). Ceci contraint le réseau à apprendre des motifs plus généraux et robustes. [51]

Figure 11 - Schéma de la fonction de la couche Dropout



(Inspiré de [51])

1.3.2.3 Prétraitement des données pour les CNN

Alors que les réseaux de neurones convolutifs (CNN) ont été initialement conçus pour traiter directement les images, ils offrent également la flexibilité de traiter d'autres types de données, comme celles stockées au format CSV. Toutefois, une telle flexibilité ne vient pas sans défis. En effet, un fichier CSV peut abriter une diversité de types de données : des chaînes de caractères (strings), des entiers (integers), et des nombres réels (floats). Néanmoins, pour assurer un fonctionnement optimal, un CNN requiert que ses données d'entrée soient sous forme de réels. D'où la nécessité d'appliquer une série de prétraitements. [41]

À partir de notre jeu de données sources au format CSV, voici les étapes essentielles de prétraitement à suivre [41] :

- **Sélectionner les attributs pertinents**
- **Encodage des chaînes de caractère**
    - **Label Encoder :**
      Chaque valeur unique d'une colonne est convertie en un nombre entier. Par exemple, les valeurs "DoS", "Worms", et "Shellcode" pourraient être représentées respectivement par les chiffres 0, 1, et 2.
    - **One-hot Encoder :**
      Chaque valeur unique d'une colonne est convertie en une nouvelle colonne dédiée. Par exemple, "DoS", "Worms", et "Shellcode" donneraient naissance à trois nouvelles colonnes, avec des encodages respectifs de [1, 0, 0], [0, 1, 0] et [0, 0, 1].
    - **Label Binarizer :**
      C'est la combinaison du label Encoding et du One-Hot Encoding [52]
- **Convertir les nombres en valeur réels**
- **Normalisation des données :**
  La normalisation des données permet de mettre à l'échelle les données. Ceci permet à chaque donnée d'avoir un poids équivalent. Par exemple, lorsqu'on compare des attributs tels que l'âge et le salaire, la normalisation empêche le salaire, généralement plus élevé, d'exercer une influence disproportionnée sur le modèle.
- **Division des données :**
  Il est essentiel de diviser le jeu de données en plusieurs sous-ensembles. Habituellement, cela inclut un ensemble d'entraînement pour former le modèle et un ensemble de tests pour évaluer sa performance. [41]

1.3.2.4 Technique d'optimisation

Ces techniques sont toutes utilisées pour ajuster et affiner les paramètres d'un réseau de neurones afin de minimiser l'erreur de prédiction. C'est grâce à ces techniques que l'on réussit à obtenir de bonne performance.

La technique du gradient descente et de la descente de gradient stochastique sont disponibles en annexe :

- **Annexe 3 : La descente de gradient**
- **Annexe 4 : La descente de gradient stochastique (SGD)**

L'algorithme d'optimisation Adam :

L'algorithme d'optimisation Adam est une variante sophistiquée de la descente de gradient stochastique. Il est fréquemment utilisé dans le domaine du deep learning, en particulier pour des applications en vision par ordinateur et en traitement du langage naturel. [56]

Adam optimise le taux d'apprentissage de chaque paramètre en se basant à la fois sur le premier moment (la moyenne historique des gradients) et le deuxième moment (la variance historique des gradients). De cette manière, Adam est capable d'ajuster de manière adaptative les mises à jour des paramètres. Cela lui permet de trouver rapidement le point le plus bas de la fonction de coût. De plus, il est robuste face aux changements d'échelle des gradients, garantissant ainsi une performance stable même lorsque les gradients présentent des amplitudes variées. [57]

2 Problématique

En quoi les systèmes de détection d'intrusion réseau peuvent-ils utiliser le Machine Learning ?

La sécurité des systèmes informatiques est un défi devenu majeur. Il y a énormément de technique différente afin de s'infiltrer dans les systèmes et accéder à des informations sensibles. Les systèmes de détection IPS/IDS sont des outils essentiels pour prévenir et détecter les cyberattaques. Cependant, les détections d'intrusion basées sur des signatures ou des anomalies peuvent se révéler limitées face à la complexité et l'évolution des attaques.

Dans ce contexte, le Machine Learning (ML) offre de nouvelles possibilités dans la détection d'infraction réseau. Le ML, et plus spécifiquement le Deep Learning, permet l'apprentissage à partir de grandes quantités de données, capable de déceler des schémas complexes que les approches traditionnelles pourraient manquer. De plus, ces techniques peuvent s'adapter aux nouvelles menaces grâce à leur capacité d'apprentissage continu.

Toutefois, l'application du ML dans le domaine de la détection d'intrusion ne va pas sans défis. L'un des obstacles majeurs souvent rencontrés dans les projets de ML est la disponibilité de jeux de données de qualité pour l'entraînement des modèles. Dans le cas des IDS/IPS, cela signifie disposer de données de trafic réseau suffisamment variées et représentatives, incluant à la fois des comportements normaux et des exemples de différents types d'attaques.

Face à ces défis, le problème central de notre travail sera d'établir comment les systèmes de détection d'intrusion réseau peuvent efficacement utiliser le Machine Learning. Quelles sont les approches de ML qui s'adaptent le mieux à ce contexte ? Enfin, comment pouvons-nous évaluer de manière fiable l'efficacité d'un modèle basé sur le ML dans la détection d'intrusion réseau, et quels sont les paramètres susceptibles d'optimiser les performances de ce modèle ?

3 Méthodologie

Afin de répondre de manière adéquate aux interrogations posées, notre démarche s'appuie sur une méthodologie concrète ancrée dans l'expérimentation.

Construction du modèle :
Notre ambition est de concevoir un modèle de détection d'intrusion basé sur le Deep Learning. Plus précisément, nous avons opté pour un réseau neuronal convolutif (CNN). La pertinence de ce choix découle de l'efficacité prouvée des CNN dans des travaux expérimentaux similaires. Notre objectif est de développer un modèle capable de classer la nature des attaques conformément à ce qui a été présenté dans notre revue de littérature.

Jeu de données :
Nous allons nous appuyer sur le jeu de données UNSW-NB15[1] pour l'entraînement de notre modèle. Ce dataset englobe un large éventail d'attaques. Il englobe tous les types d'attaques mentionnés dans notre étude, tout en intégrant également des trafics considérés comme légitimes. Cependant, avant d'être exploitable, ce jeu de données nécessite un traitement préalable afin de l'adapter à notre CNN. Notre ambition est de le transformer en un fichier csv optimisé pour notre architecture. Voici, de manière séquentielle, les étapes que nous comptons suivre pour préparer ce fichier :

- Rectification des anomalies et des erreurs présentes dans le dataset d'origine.
- Suppression des colonnes non pertinente.
- Afin d'éviter un déséquilibre dans notre dataset qui pourrait biaiser l'apprentissage du réseau neuronal en faveur des trafics dominants, nous avons procédé à un rééquilibrage. Nous avons ainsi ajusté le volume de trafic légitime pour qu'il corresponde à deux tiers du nombre d'attaques de la catégorie « Generic ». Cette démarche assure une meilleure représentativité des différentes classes dans le dataset
- Encodage des variables catégorielles à l'aide du label encoder.
- Normalisation des données pour qu'elles se situent dans l'intervalle [0,1].
- Quantification des données entre 0 et 255 pour faciliter leur conversion en format "image".
- Application du OneHotEncoding sur la colonne représentant la catégorie d'attaque

Une fois ces étapes achevées, notre fichier CSV est fin prêt pour l'entraînement.

[1] Afin d'avoir les autorisations d'utiliser ce dataset, nous devons citer ces publications [58, 59, 60, 61]

Mise en œuvre du modèle :
Après la préparation des données, la prochaine étape concerne la conception de notre CNN en utilisant la librairie Keras. Lors de l'import du fichier csv, nous effectuerons une transformation afin d'obtenir des images de taille 8x8 à partir de nos données.

Conception d'un réseau neuronal convolutif (CNN) :
Un réseau neuronal convolutif (CNN) est spécifiquement conçu pour le traitement des images. Nous allons mettre en place plusieurs CNN dérivant de cette structure basique.

1. **Bloc 1** (peut se répéter un certain nombre de fois) :
    - **Couches de Convolution :**
      Les images entrent dans les couches de convolution, où des filtres s'appliquent, créant des sorties appelées "feature maps."
        - **Fonction d'Activation :**
          Ces feature maps passeront par une fonction d'activation, comme la fonction ReLU, afin d'ajouter de la non-linéarité.
    - **Couche de Pooling :**
      Elle permet de réduire les dimensions tout en conservant les informations les plus importantes. Étant donné que nous avons de petites images, ce point pourrait être problématique, et nous envisagerons donc plusieurs tests d'architecture pour y remédier.
    - **(Optionnel) Normalisation et Dropout :**
      Nous pourrions également intégrer des couches de normalisation et de dropout pour améliorer la performance du modèle.
2. **Couche Entièrement Connectée :**
    - Ce sont des couches ou chaque neurone est connecté à tous les neurones de la couche précédente et de la couche suivante.
    - La fonction d'activation softmax peut être utilisée dans la dernière couche entièrement connectée pour obtenir une probabilité pour chaque classe.

En résumé, à la fin de cette architecture, nous obtenons une probabilité associée à chaque image correspondant à une classe spécifique. Le choix du nombre de blocs, des fonctions d'activation, de pooling, et de la structure de la couche entièrement connectée sera guidé par des expérimentations afin de trouver l'architecture optimale pour notre cas d'utilisation.

Par la suite, nous passerons à la phase d'entraînement et de test. Pour la validation, nous privilégierons la méthode de "stratified validation", appréciée pour sa capacité à préserver une répartition équilibrée des classes. En comparaison, bien que la validation croisée soit utile, elle s'avérerait trop exigeante en ressources pour notre cas d'usage.

Le choix du modèle optimal sera guidé par une exploration progressive : à chaque étape, un modèle sera retenu pour des ajustements ultérieurs. Nos critères d'évaluation et de comparaison des différents modèles incluent : la précision (accuracy), le taux d'erreur (loss), la complexité (nombre de paramètres) et le temps d'entraînement.

Nous procéderons à une série de tests rigoureux pour identifier les paramètres optimaux garantissant la performance optimale de notre modèle.

1. **Exploration de diverse architecture :**
   Dans le cadre de ce travail, il est primordial de définir l'architecture la plus appropriée pour résoudre notre problème spécifique. Ainsi, nous envisagerons d'examiner 2 architectures reprises de projet github similaire[2], ainsi que 3 autres architectures notables, à savoir « DenseNet » (simplifiée), « GoogleNet » (simplifiée), et « MiniVGG ». L'évaluation comparative de ces architectures permettra d'établir qu'elle est la meilleure architecture pour notre problème.

2. **Différents essais sur les couches de profondeur et les transitions :**
   La seconde phase consistera en une évaluation méthodique de l'impact du nombre de couches de profondeur sur les performances du modèle. Nous envisagerons des modèles composés de 1 à 5 couches de profondeur. Parallèlement, nous étudierons l'efficacité potentielle des couches de transition pour améliorer la performance du modèle. À cet égard, nous expérimenterons également des modèles de 1 à 5 couches intégrant des couches de transition.

   Il convient de souligner que ces couches de transition sont généralement composées d'une convolution avec un filtre de dimension 1x1, suivie d'une activation ReLU, et d'une normalisation. Bien que l'intégration d'une opération de pooling soit habituelle dans les couches de transition, nous optons pour son exclusion, compte tenu de la petite taille des images (8x8).

   L'objectif de ce test vise à déterminer le nombre optimal de couches pour le modèle tout en évaluant si l'inclusion de transitions peut diminuer le nombre de paramètres.

3. **Essai de plusieurs couches interconnectées :**
   Dans cette phase, nous envisagerons d'analyser les effets de l'interconnexion de différentes couches au sein du modèle. Les configurations à tester comprendront la mise en œuvre de 1, 2 et 3 couches interconnectées, avec diverses tailles de neurones et l'incorporation de couche de dropout.

   Le choix de ces différentes combinaisons permettra d'évaluer si l'ajout de couche interconnectée aurait un impact positif sur la classification

4. **Test sur plusieurs tailles de lots (batch_size) :**
   Le batch size est un hyperparamètre crucial dans l'entraînement des réseaux neuronaux. Il détermine le nombre d'exemples d'entraînement traités avant la mise à jour des poids du modèle.

   Dans le cadre de ce test, nous allons explorer l'effet de six tailles de lots différentes sur la performance de l'entraînement. Ces tailles de lots comprennent : 32, 64, 128, 256, 512 et 1024.

   L'objectif de cette phase de test est précis : déterminer la taille de lot qui optimise la performance de notre modèle.

[2] Projet GitHub similaire où nous avons récupéré leur architecture : [62, 63]

Après avoir mené ces tests approfondis, nous identifierons les paramètres optimaux qui conviennent à notre modèle. Cette démarche nous fournira les informations nécessaires pour concevoir un modèle parfaitement adapté à notre contexte. Le modèle final sera soumis à 50 cycles d'entraînement et sera ensuite sauvegardé. Nous évaluerons les résultats du modèle finalisé à l'aide des métriques suivantes : accuracy, précision, recall et le F1 Score, ainsi que la matrice de confusion. Ces métriques nous permettront d'obtenir un aperçu précis des résultats obtenus, et de comprendre de manière détaillée la performance de notre modèle dans la tâche de classification.

4 Réalisation

L'intégralité de notre projet, centré sur la détection d'intrusion réseau par l'apprentissage automatique, est disponible sur notre dépôt GitHub. Le travail est structuré en deux parties principales, chacune contenue dans un fichier Jupyter Notebook distinct. Voici un aperçu de ces fichiers :

- **Préparation des données (prepare_data_CNN.ipynb) :**
- **Mise en place du CNN (CNN_Multiclass.ipynb) :**

Vous pouvez accéder au repository github : https://github.com/TWAAXOne/Network-intrusion-detection-with-machine-learning

Ce dépôt offre une vision complète et transparente de notre démarche, des choix méthodologiques aux expérimentations réalisées. Cela permet à toute personne intéressée par notre projet de comprendre en détail notre approche, et éventuellement de contribuer ou d'adapter notre travail à ses propres besoins.

4.1 Création du fichier csv

La préparation des données est une étape essentielle pour le bon fonctionnement d'un réseau de neurones convolutif (CNN). Dans le cadre de notre projet, nous utilisons le dataset UNSW-NB15, qui nécessite une compréhension approfondie pour une utilisation efficace.

Une explication détaillée du dataset UNSW-NB15, y compris sa structure, ses caractéristiques et son importance dans le contexte de la détection d'intrusion, est fournie en **Annexe 5 : Explication du dataset UNSW-NB15**

En partant des 4 fichiers CSV initiaux, plusieurs améliorations et modifications ont été nécessaires pour obtenir un jeu de données compatible avec notre CNN. Pour ceci, j'ai créé un script intitulé « prepare_data_CNN ». Voici les étapes suivies :

1. **Fusion des fichiers CSV**
   Les quatre fichiers CSV existants ont été fusionnés en un seul tableau pour avoir une vue d'ensemble des données.

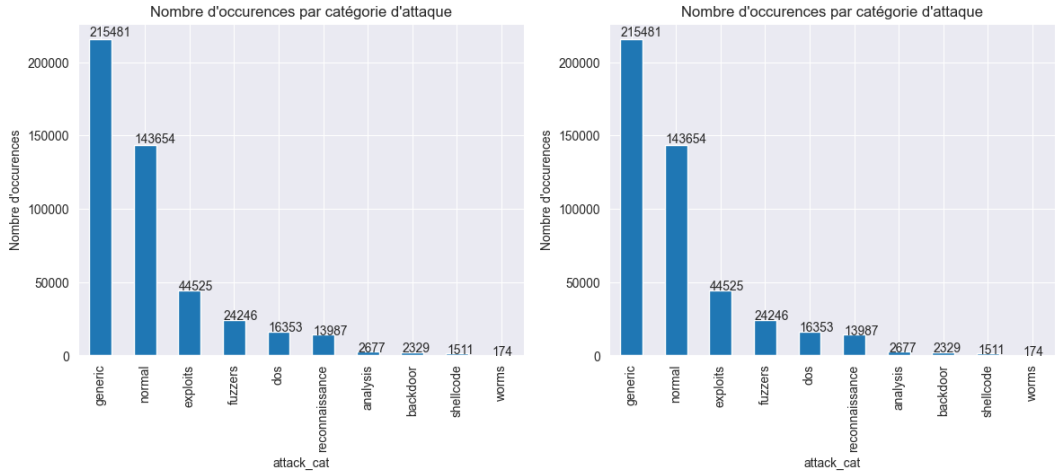
2. **Récupération des noms de colonnes**
   Une liste des noms des colonnes a été extraite pour faciliter les opérations de traitement ultérieures.

3. **Gestion des valeurs manquantes et des anomalies :**
   - Les entrées vides de la colonne « is_ftp_login » ont été remplacées par 0. De plus, toutes les valeurs de cette colonne supérieures à 1 ont été transformées en 1.
   - Les valeurs null de « ct_flw_http_mthd » ont été remplacées par 0.
   - Dans attack_cat, les entrées nulles ont été changées en « normal » et certaines incohérences, comme les doublons comme Backdoor et Backdoors, ont été corrigées.

4. **Suppression de certaines colonnes**
   Les colonnes telles que « srcip » (adresse IP source), « sport » (port source), « dstip » (adresse IP de destination), et « dsport » (port de destination) ont été éliminées. Ces attributs, relatifs à l'adresse IP et aux ports, ne contribuent pas de manière significative à la tâche de classification envisagée. Ils pourraient introduire un bruit inutile.

5. **Suppression d'une grande partie des attaques « normal »**
   La configuration initiale du dataset est illustrée par le graphique de gauche. Cependant, une telle répartition hétérogène risque de perturber la phase d'entraînement et de biaiser les résultats. Afin d'harmoniser la distribution, nous avons opté pour une réduction significative des attaques de type « normal ». Notre objectif était d'aligner leur nombre à 2/3 de celui des attaques « génériques », soit un total de 143'654 occurrences. Cette intervention nous a permis d'obtenir un dataset plus équilibré, comme le montre le graphique de droite.

Figure 12 - Comparaison des Proportions des Classes dans le Dataset avant et après la suppression



6. **Encodage des valeurs catégorielles**
   Les colonnes « proto », « state », et « service » contiennent des données catégorielles qui doivent être transformées en valeurs numériques pour être traitées par le modèle. Afin d'effectuer cette conversion, nous avons utilisé la classe LabelEncoder de la bibliothèque scikit-learn. Grâce à la fonction fit_transform, ces colonnes ont été encodées en valeurs numériques.

7. **Conversion en float**
   L'ensemble des attributs, à l'exception de « attack_cat », a été converti en type float. Cette étape est cruciale, car les CNN traitent exclusivement des données sous forme numérique.

8. **Normalisation des données**
   Pour assurer l'uniformité des différentes variables, nous avons normalisé les données afin qu'elles se situent entre 0 et 1. Utilisant la méthode Min-MaxScaler de scikit-learn, cette normalisation a été appliquée à toutes les colonnes, sauf celles contenant des données catégorielles, telles que "attack_cat", "label", et trois autres colonnes catégorielles. Cette étape permet d'éviter les biais causés par les disparités d'échelle entre différentes caractéristiques.

9. **Quantification**
   Afin de transformer chaque ligne de notre CSV en une "image", nous avons procédé à une quantification des valeurs. Ces dernières ont été ajustées pour s'échelonner de 0 à 255, permettant à chaque pixel de l'image de refléter une nuance allant du noir (0) au blanc (255). Pour faciliter cette conversion, nous avons conçu une fonction de quantification dédiée. J'ai également utilisé la méthode Min-MaxScaler de skit-learn. Elle a été mise en œuvre sur les mêmes colonnes que celles traitées par la normalisation.

10. **Encodage one-hot**
    La colonne "attack_cat" a été transformée à l'aide de l'encodage one-hot, créant ainsi plusieurs colonnes binaires. À la suite de cette transformation, nous avons retiré la colonne originale du jeu de données. Cette étape était essentielle, car elle permet de convertir les catégories en une forme que les modèles algorithmiques peuvent mieux exploiter. Pour réaliser cet encodage, j'ai fait appel à la fonction get_dummies de pandas, spécialement conçue pour effectuer du one-hot encoding.

11. **Sauvegarde du jeu de données**
    Les données préparées ont été enregistrées dans un fichier CSV pour une utilisation ultérieure.

Après avoir sauvegardé le fichier CSV, les données sont désormais prêtes pour l'entraînement de notre modèle. Initialement, le fichier contenait 2'540'044 lignes avec 49 attributs (features). À la suite de notre transformation, il compte maintenant 464'939 lignes et de 54 attributs.

4.2 Étapes préliminaires à la conception des modèles

Pour initier la conception de nos modèles, nous commençons par importer notre fichier CSV dans un autre script, que nous avons nommé « CNN_Multiclass.ipynb ». La première étape consiste à extraire les dix types d'attaques (y compris les attaques normales) et à les stocker dans la variable 'y'. Par la suite, nous éliminons la colonne « label » qui sert principalement à indiquer si une entrée est une attaque ou non, car elle ne sera pas pertinente pour notre modèle.

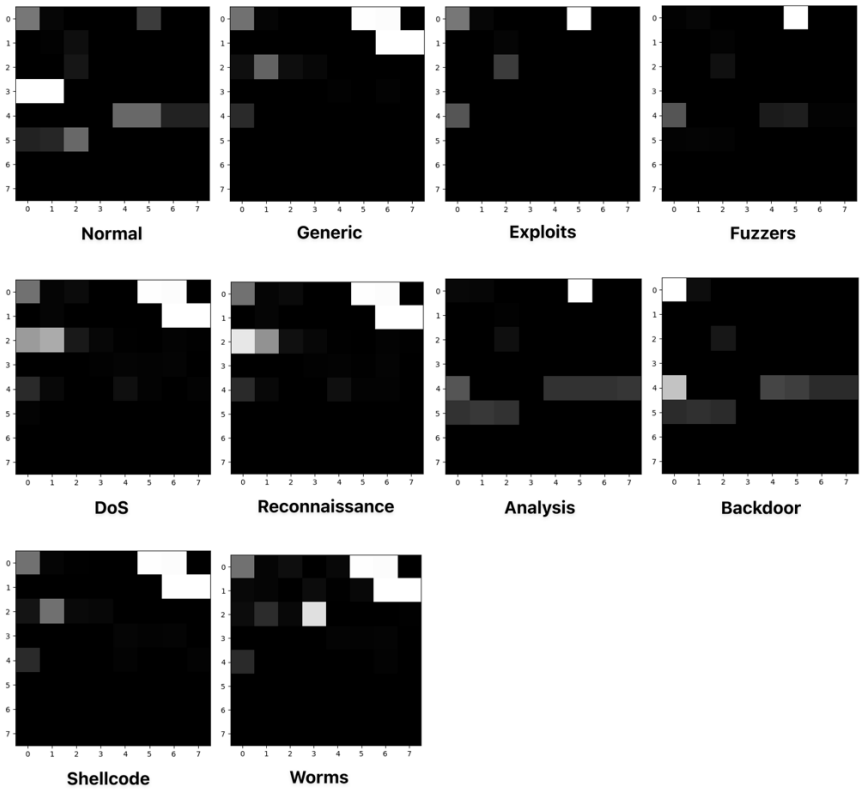
Le défi majeur auquel nous sommes confrontés est la transformation de chaque ligne de notre fichier CSV en une image. Concrètement, notre objectif est de créer des images de dimensions 8x8 pixels. Toutefois, un obstacle se présente : chaque ligne de notre fichier ne contient actuellement que 43 valeurs, alors que pour obtenir une image 8x8, il nous faut 64 valeurs. Pour pallier ce problème, nous allons compléter chaque ligne avec 21 valeurs supplémentaires, toutes égales à zéro, pour atteindre les 64 valeurs requises. Le code que nous utilisons permet d'ajouter ces 21 colonnes de zéros à la fin de chaque entrée. Ainsi, '**byte_images'** correspond à un tableau numpy composé de 464'937 entrées, chacune contenant un tableau de 64 valeurs.

Figure 13 - Code de transformation en un tableau de 64 valeurs

```
1. byte_images = np.pad(data.to_numpy(), ((0, 0), (0, 21)), 'constant')
```

Pour visualiser les images qui serviront à l'entraînement de notre réseau, nous avons conçu une fonction nommée '**plot_single_image'**. Cette fonction prend une entrée de '**byte_images'** et, grâce à ses 64 valeurs, la redimensionne en une matrice 8x8, simulant ainsi la structure d'une image. La fonction affiche ensuite l'image sous forme de niveaux de gris. Ainsi, nous pouvons afficher les images de plusieurs catégories d'attaque.

Figure 14 - Images des catégories d'attaques



L'utilisation des réseaux neuronaux pour la détection d'intrusions dans les réseaux informatiques

Avant de séparer notre jeu de données en ensembles d'entraînement et de tests, nous devons restructurer l'entrée « x » en une matrice 4D. Cette matrice aura une dimension de 464'939 x 8 x 8 x 1. Ainsi, nous disposerons de 464'939 échantillons, chacun étant une image de taille 8x8 avec une seule couche de profondeur.

Figure 15 - Code de restructuration de x en une matrice 4D (464'949 x 8 x 8 x 1)

```
1. x = byte_images.reshape(data.shape[0], 8, 8, 1)
```

Séparation des données
Plutôt que d'opter pour une validation croisée, méthode souvent gourmande en temps, nous avons choisi d'utiliser la validation stratifiée. Cette méthode nous permet de garantir que la distribution des classes dans les ensembles d'entraînement et de tests reflète la distribution originale de l'ensemble de données complet.

Dans ce cadre, nous avons alloué 1/3 des données pour les tests et le reste pour l'entraînement. Pour réaliser cette séparation, nous avons appliqué une séparation stratifiée sur nos données x (qui contient toutes les colonnes sauf la cible) et y (la colonne cible). De cette façon, nous avons obtenu quatre sous-ensembles : x_train et x_test pour les caractéristiques d'entraînement, ainsi que y_train et y_test pour les tests.

4.2.1 Construction et optimisation des paramètres des modèles

Pour optimiser notre modèle, une démarche méthodique est essentielle. Lors de la création de nos couches de convolution, nous appliquerons une stratégie de "padding" avec la valeur "same". Cette approche garantit que la taille des features maps reste constante à travers les couches de convolution, empêchant ainsi la réduction excessive de sa taille. Cela assure que la carte des caractéristiques à la fin de chaque couche de convolution sera égale à la taille de l'image d'entrée, préservant ainsi les informations spatiales importantes.

De plus, nous utiliserons la fonction d'activation "SoftMax" en sortie de la couche dense, car elle est spécifiquement conçue pour la prédiction dans des scénarios de classification multiclasse. Si notre objectif avait été de classer les événements en deux catégories seulement, tels qu'attaque ou non-attaque (classification binaire), la fonction d'activation "sigmoid" aurait été plus adaptée, et la fonction de perte appropriée aurait été "binary_crossentropy". Dans le contexte actuel de classification multi classe, nous avons choisi d'adopter l'optimiseur "adam" et la fonction de perte "categorical_crossentropy".

Nous avons choisi d'utiliser l'optimiseur "adam" et la fonction de perte "categorical_crossentropy", car ces méthodes sont largement répandues dans des projets similaires. L'avantage d'utiliser "adam" réside dans sa rapidité et son efficacité lors de l'entraînement, le rendant particulièrement adapté à notre contexte. Quant à "categorical_crossentropy", elle est fréquemment utilisée dans la classification multiclasse. Cette fonction de perte mesure la différence entre les probabilités de sortie prédites et les étiquettes réelles, fournissant une métrique précise pour évaluer la performance du modèle dans notre cas de classification à plusieurs catégories.

4.2.1.1 Expérimentation 1 : Exploration de diverses architectures

Notre première mission est de définir l'architecture la plus pertinente pour notre réseau de neurones. Pour ce faire, nous envisageons d'entraîner le même modèle 20 fois avec un ensemble de données constant et un batch size fixé à 128. Dans cette démarche, j'ai sélectionné cinq architectures distinctes pour évaluation :

- **ClassicConvNet :**
  Cette architecture, inspirée d'un projet similaire trouvé sur GitHub [62], est une représentation typique d'un CNN. Elle comprend trois blocs de convolution, dont le nombre de filtres augmente progressivement, suivi respectivement par une couche de maxpooling. Enfin, le réseau se termine par deux couches denses et une couche de dropout.

  Figure 16 - Code typique d'un CNN

```
input_img = Input(shape=(8, 8, 1))  # Définition de la taille des images
block_1 = Conv2D(64, (3, 3), activation='relu', padding='same')(input_img)
block_1 = Conv2D(64, (3, 3), activation='relu', padding='same')(block_1)
block_1 = MaxPooling2D(pool_size=(2, 2))(block_1)
block_1 = Conv2D(128, (3, 3), activation='relu', padding='same')(block_1)
block_1 = Conv2D(128, (3, 3), activation='relu', padding='same')(block_1)
block_1 = Conv2D(128, (3, 3), activation='relu', padding='same')(block_1)
block_1 = MaxPooling2D(pool_size=(2, 2))(block_1)
block_1 = Conv2D(256, (3, 3), activation='relu', padding='same')(block_1)
block_1 = Conv2D(256, (3, 3), activation='relu', padding='same')(block_1)
block_1 = Conv2D(256, (3, 3), activation='relu', padding='same')(block_1)
block_1 = MaxPooling2D(pool_size=(2, 2))(block_1)
output = Flatten()(block_1)
# Bloc 2
output = Dense(100, kernel_initializer='normal', activation='relu')(output)
output = Dropout(0.5)(output)
output = Dense(20, kernel_initializer='normal', activation='relu')(output)
out = Dense(nb_classes, kernel_initializer='normal', activation='softmax')(out-
put)
```

- **NormedConvNet :**
  Cette architecture est également inspirée d'un projet similaire trouvé sur GitHub [63]. Elle est constituée de deux couches convolution de 32 filtres chacune, chacune précédée d'une normalisation et d'une activation ReLU. Après la convolution, nous passons deux couches denses. Son avantage c'est qu'il y a peu de couches de convolution, ce qui rend son architecture très simple et les couches de normalisation devrait accélérer son apprentissage.

- **MiniVGGNet**
  Ce modèle est une version épurée du célèbre réseau VGGNet. Il se compose de deux blocs de convolution, chacun suivi d'une normalisation par lots, facilitant ainsi une convergence plus rapide lors de l'entraînement. Après ces blocs, le réseau intègre une couche de MaxPooling pour réduire les dimensions, puis un dropout pour régulariser et prévenir le surapprentissage. Le modèle se conclut par une couche dense, accompagnée d'une nouvelle normalisation et d'un dropout. Grâce à sa simplicité, cette architecture assure un entraînement efficace et rapide.

- **GoogleNet (version simplifiée) :**
  L'architecture GoogleNet, dans sa version simplifiée présentée ici, initialement développée par Google. Cette version se distingue par l'utilisation de plusieurs tours fonctionnant en parallèle, chacun avec des filtres de tailles variées, avant de combiner leurs résultats et de passer par une couche dense avec la classification softmax.

- **DenseNet (version simplifiée)** :
  L'architecture DenseNet, dans cette version simplifiée, se caractérise par ses connexions denses. Au lieu de connecter chaque couche uniquement à la suivante (comme dans une architecture classique), chaque couche reçoit en entrée toutes les sorties des couches précédentes et transmet sa propre sortie à chaque couche suivante. Cette architecture a pour caractéristique d'avoir moins de paramètres.

Figure 17 - Code d'un CNN d'architecture DenseNet (simplifié)

```
x = Conv2D(32, (3, 3), padding='same', activation='relu')(input_img)
x = BatchNormalization()(x)
x1 = Conv2D(32, (3, 3), padding='same', activation='relu')(x)
x1 = BatchNormalization()(x1)
concat1 = concatenate([x, x1]) # concaténation de la sortie de la couche 1 et 2
x2 = Conv2D(32, (3, 3), padding='same', activation='relu')(concat1)
x2 = BatchNormalization()(x2)
concat2 = concatenate([x, x1, x2]) # concaténation des sorties des 3 couches
output = Flatten()(concat2)
out = Dense(nb_classes, activation='softmax')(output)
```

Chaque architecture est entraînée de la même façon à l'aide de ce bloc de code. Le terme "net" représente le numéro de l'architecture, "input_img" correspond aux images d'entrée de taille 8x8x1, et "out" est la sortie obtenue après avoir traversé chaque couche. Le modèle est compilé en utilisant l'optimiseur "adam" et la fonction de perte "entropie croisée catégorique". L'entraînement est effectué sur les données x_train et y_train, et à la fin de chaque époque, le modèle est évalué sur un ensemble de données de validation (x_test, y_test), spécifié par l'argument validation_data.

Figure 18 - Code pour la compilation du modèle

```
model[net] = Model(inputs=input_img, outputs=out)
modèle[net].summary()
modèle[net].compile(optimizer="adam", loss="categorical_crossentropy", metrics=["accuracy"])
history[net] = model[net].fit(x_train, y_train, validation_data=(x_test, y_test), verbose=1, batch_size=batch_size, epochs=epochs)
```

Après l’entraînement de notre modèle, voici les résultats que l’on obtient :

Figure 19 - Graphique comparatif des taux de précision entre différentes architectures de CNN

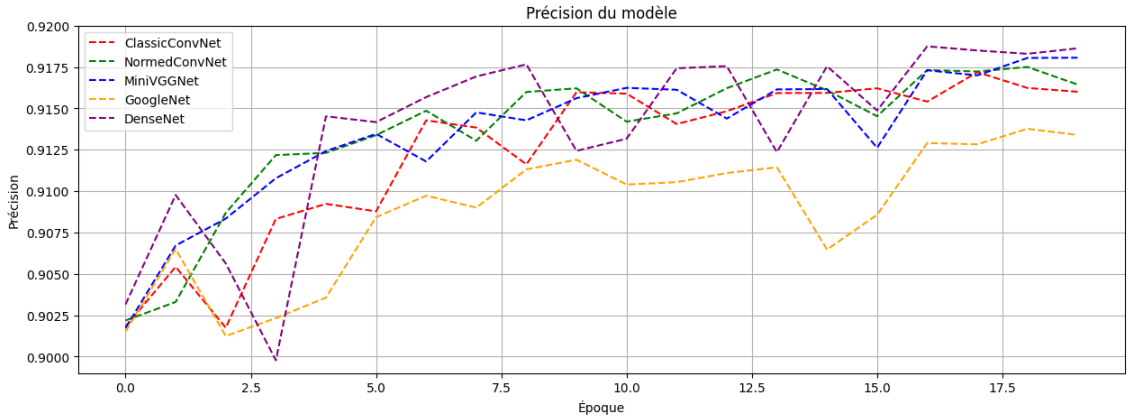


Figure 20 - Tableau comparatif des différentes architectures d'un CNN

Nom	Accuracy	Loss	Nombre de paramètres	Temps d’entraînement
DenseNet	0.918628	0.220312	89’866	331.89
MiniVGGNet	0.918061	0.217676	279’658	332.09
NormedConvNet	0.916441	0.223323	273’262	252.82
ClassicConvNet	0.916004	0.225155	1'909’850	395.37
GoogleNet	0.913394	0.235815	262’666	278.48

Après avoir testé nos cinq différentes architectures, l'analyse des résultats montre des distinctions notables. GoogleNet se démarque par ses performances inférieures, tandis que les quatre autres modèles offrent des performances assez proches. Mais l’ensemble des architectures avoisine des précisions proches de 91%.

DenseNet se distingue comme étant le modèle le plus performant en termes de précision, tout en ayant le plus faible nombre de paramètres. MiniVGGNet et NormedConvNet sont assez similaires en performance, bien que le NormedConvNet se démarque par son temps d'entraînement plus rapide. Cependant, le ClassicConvNet, bien qu'ayant des résultats respectables, présente un nombre de paramètres nettement supérieur et requiert plus de temps pour l'entraînement.

Compte tenu de ces observations, nous choisirons DenseNet comme modèle de référence pour les expérimentations ultérieures.

4.2.1.2 Expérimentation 2 : Différents essais sur les couches de profondeur et les transitions

Notre objectif ici est d'évaluer l'impact de la profondeur du modèle, c'est-à-dire le nombre de couches, sur la qualité de ses prédictions. De plus, nous examinerons comment l'introduction de couches de transition peut éventuellement optimiser les performances du modèle. L'idée sous-jacente est de rendre le modèle plus complexe tout en simplifiant sa structure.

Dans cette expérimentation, j'ai élaboré cinq variantes du modèle DenseNet. Chaque modèle se distingue par sa profondeur : le premier modèle est composé d'une seule couche de convolution, tandis que le deuxième en a deux, et ainsi de suite pour les suivants. Exemple du modèle à deux couches :

Figure 21 - Code d'un DenseNet à deux couches de convolution

```
# première couche de convolution
x1 = Conv2D(32, (3, 3), padding='same', activation='relu')(input_img)
x1 = BatchNormalization()(x1)
# deuxième couche de convolution
x2 = Conv2D(32, (3, 3), padding='same', activation='relu')(x1)
x2 = BatchNormalization()(x2)
concat2 = concatenate([x1, x2])
output = Flatten()(concat2)
out = Dense(nb_classes, activation='softmax')(output)
```

Ensuite, j'ai conçu cinq autres variantes de DenseNet similaires à celles précédemment présentées. La principale distinction est l'ajout de couches de transition après chaque opération de concaténation. Pour illustrer, nous présentons ici le modèle à 4 couches de convolution, qui intègre une couche de convolution suivie d'une couche de transition.

Figure 22 - Code d'un DenseNet à quatre couches de convolution avec des couches de transition

```
def transition_layer(x, num_filters):
    x = Conv2D(num_filters, (1, 1), padding='same', activation='relu')(x)
    x = BatchNormalization()(x)
    return x
```

```
# première couche de convolution
x1 = Conv2D(32, (3, 3), padding='same', activation='relu')(input_img)
x1 = BatchNormalization()(x1)
# deuxième couche de convolution
x2 = Conv2D(32, (3, 3), padding='same', activation='relu')(x1)
x2 = BatchNormalization()(x2)
concat2 = concatenate([x1, x2])
concat2 = transition_layer(concat2, 32)  # couche de transition
# troisième couche de convolution
x3 = Conv2D(32, (3, 3), padding='same', activation='relu')(concat2)
x3 = BatchNormalization()(x3)
concat3 = concatenate([x1, x2, x3])
concat3 = transition_layer(concat3, 32)  # couche de transition
# quatrième couche de convolution
x4 = Conv2D(32, (3, 3), padding='same', activation='relu')(concat3)
x4 = BatchNormalization()(x4)
concat4 = concatenate([x1, x2, x3, x4])
concat4 = transition_layer(concat4, 32)  # couche de transition
output = Flatten()(concat4)
out = Dense(nb_classes, activation='softmax')(output)
```

Après l'entraînement de nos modèles, voici ce que l'on obtient :

Figure 23 - Tableau comparatif des variantes d'un DenseNet avec différentes couches de convolution

Nom	Accuracy	Loss	Nombre de paramètres	Temps d'entraînement
3_blocks_without	0.919015	0.221782	89'866	254
4_blocks_without	0.916595	0.219597	138'154	292
5_blocks_without	0.914988	0.220117	195'658	325
2_blocks_without	0.913001	0.230009	50'794'	220
1_blocks_without	0.909175	0.248807	20'938	191

Figure 24 - Tableau comparatif des variantes d'un DenseNet avec différentes couches de convolution avec couche de transition

Nom	Accuracy	Loss	Nombre de paramètres	Temps d'entraînement
4_blocks_with	0.920456	0.216813	60'074	461
5_blocks_with	0.920233	0.220955	74'762	518
3_blocks_with	0.918800	0.225745	46'410	393
2_blocks_with	0.916864	0.226845	33'770	311
1_blocks_with	0.914003	0.252406	22'154	272

Les résultats démontrent que l'accuracy et la perte sont relativement similaires entre les différents modèles, bien que le modèle à une couche soit nettement inférieur aux autres. Ce graphique révèle que le nombre de couches de convolution est lié de manière significative au nombre de paramètres du modèle. Plus intéressant encore, l'ajout de couches de transition permet de réduire de façon substantielle le nombre de paramètres tout en augmentant légèrement la précision et du loss, bien que cela entraîne une augmentation du temps d'entraînement.

Bien que les performances en termes de précision soient très proches pour tous les modèles, l'évaluation du nombre de paramètres et du temps d'entraînement joue un rôle crucial dans la sélection du modèle optimal. Le modèle à 4 blocs avec couches de transition émerge comme le choix privilégié pour la prochaine phase de tests, offrant un équilibre attrayant entre précision et efficacité en termes de ressources.

4.2.1.3 Expérimentation 3 : Essais de plusieurs couches interconnectées

Dans cette expérimentation, nous allons tester plusieurs architectures de couches interconnectées afin d'analyser et de comparer leur efficacité dans un contexte spécifique. Pour ceci, nous reprenons la base d'architecture de l'expérimentation 2 qui est constituée de 4 couches de convolution avec des couches de transition.

L'objectif est de déterminer quelle structure offre les meilleures performances. Les configurations testées sont les suivantes :

- 1 couche dense avec 10 neurones, destinée à la classification.
- 2 couches denses avec 32 et 10 neurones, respectivement.
- 2 couches denses avec 64 et 10 neurones.
- 3 couches denses avec 128, 64 et 10 neurones. Entre les couches denses, des "dropout" de 0.5 sont ajoutés pour prévenir le surapprentissage.
- 3 couches denses avec 128, 64 et 10 neurones, sans dropout.

Figure 25 - Code des différentes architectures de couches interconnectées

```
input_img = Input(shape=(8, 8, 1))
if net == 0: # 1 dense block
    output = denseNetModel(input_img)
    out = Dense(nb_classes, activation='softmax')(output)
if net == 1: # 2 dense blocks with 32 neurons
    output = denseNetModel(input_img)
    dense1 = Dense(32, activation='relu')(output)
    out = Dense(nb_classes, activation='softmax')(dense1)
if net == 2: # 2 dense blocks with 64 neurons
    output = denseNetModel(input_img)
    dense1 = Dense(64, activation='relu')(output)
    out = Dense(nb_classes, activation='softmax')(dense1)
if net == 3: # 3 dense blocks with 128, 64 neurons with dropout
    output = denseNetModel(input_img)
    dense1 = Dense(128, activation='relu')(output)
    dropout1 = Dropout(0.5)(dense1)
    dense2 = Dense(64, activation='relu')(dropout1)
    dropout2 = Dropout(0.5)(dense2)
    out = Dense(nb_classes, activation='softmax')(dropout2)
if net == 4: # 3 dense blocks with 128, 64 neurons without dropout
    output = denseNetModel(input_img)
    dense1 = Dense(128, activation='relu')(output)
    dense2 = Dense(64, activation='relu')(dense1)
    out = Dense(nb_classes, activation='softmax')(dense2)
```

L'expérimentation a pour but de mettre en lumière la relation entre la complexité de l'architecture et l'efficacité du modèle, en observant comment les différentes structures affectent les résultats de classification ou d'autres tâches spécifiques.

Après l’entraînement de nos modèles, voici ce que l’on obtient :

Figure 26 - Graphique de comparaison des modèles avec plusieurs couches dense

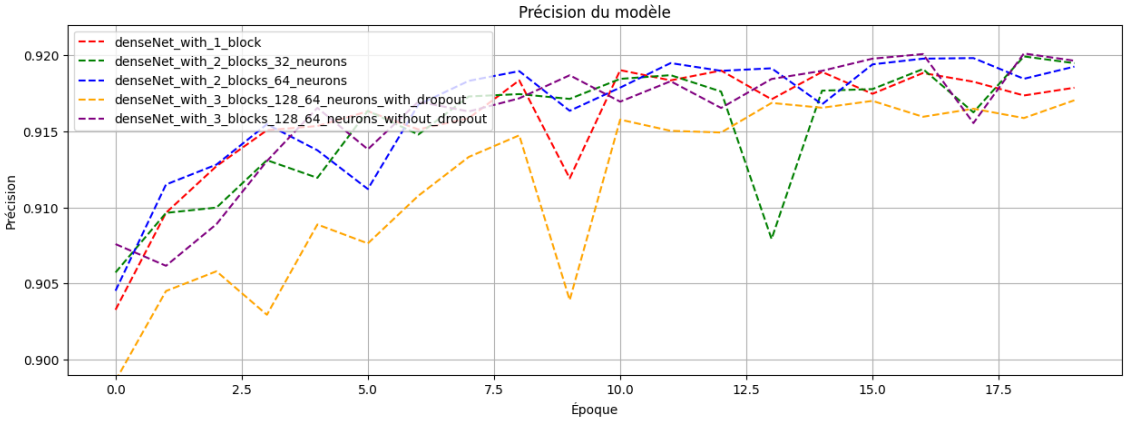


Figure 27 - Tableau de comparaison des modèles avec plusieurs couches dense

Nom	Accuracy	Loss	Nombre de paramètres	Temps d’entraînement
3 blocks 128, 64 neurons without dropout	0.919641	0.215836	309’450	400
2 blocks 32 neurons	0.919479	0.214307	104’170	382
2 blocks 64 neurons	0.919254	0.214809	170’058	381
1 block	0.916866	0.219940	58’762	381
3 blocks 128, 64 neurons with dropout	0.917040	0.226796	309’450	403

Nous pouvons observer que l'accuracy, la perte (loss) et le temps d'entraînement sont remarquablement similaires à travers tous ces modèles, et que l'ajout de couches denses supplémentaires ne parvient pas à augmenter l'accuracy ou à réduire la perte de manière significative. En revanche, cet ajout de couches a un impact considérable sur la complexité du modèle en multipliant le nombre de paramètres.

De plus, l'intégration d'une couche de dropout ne semble pas avoir un impact positif significatif ; au contraire, elle rend le modèle légèrement moins performant.

Compte tenu de ces observations, nous allons considérer le modèle avec une seule couche dense comme étant le meilleur choix. Bien que les performances, la perte et le temps d'entraînement soient relativement proches entre les différents modèles, le modèle le plus simple présent l'avantage de la simplicité. Il répond efficacement aux besoins sans ajouter de complexité inutile. En somme, ces résultats soulignent l'importance d'une conception réfléchie et de l'évaluation de multiples facteurs lors de la sélection de l'architecture d'un modèle de réseau neuronal.

4.2.1.4 Expérimentation 4 : Tests sur plusieurs tailles de lots (batch_size)

Dans cette quatrième expérimentation, nous allons mettre à l'épreuve le modèle que nous avons choisi précédemment : le DenseNet composé de quatre couches de convolution et des couches de transition, complété par une couche dense avec la fonction d'activation softmax.

Notre objectif ici est d'évaluer l'impact de différentes tailles de lots (batch sizes) sur l'entraînement du modèle. Cela nous permettra d'identifier la taille de lot optimale pour notre contexte spécifique. Les tailles de lots que nous comparerons sont les suivantes : 32, 64, 128, 256, 512 et 1024.

La taille de lot est un paramètre important dans l'entraînement d'un modèle de réseau neuronal. Elle détermine le nombre d'échantillons d'entraînement qui seront utilisés pour mettre à jour les poids du modèle à chaque itération de l'algorithme d'optimisation. Une taille de lot trop petite peut conduire à une convergence instable, tandis qu'une taille trop grande peut ralentir l'apprentissage.

Cette expérimentation nous fournira des insights précieux sur la manière dont la taille de lot influence la vitesse d'entraînement, la stabilité et la performance finale du modèle.

Après l'entraînement du modèle, voici les résultats que l'on obtient.

Figure 28 - Tableau de comparaison des différents batch size

Nom	Accuracy	Loss	Nombre de paramètres	Temps d'entraînement
256 batch_size	0.920292	0.213403	58'762	197
32 batch_size	0.919925	0.212613	58'762	1443
128 batch_size	0.918544	0.216014	58'762	373
512 batch_size	0.918305	0.215885	58'762	112
64 batch_size	0.917911	0.222639	58'762	721
1024 batch_size	0.902793	0.244030	58'762	76

Nous pouvons constater que le choix de la taille du lot (batch size) n'a pas d'effet majeur sur la précision (accuracy) ni sur la perte (loss), à l'exception notable du lot de 1024 qui affiche des performances légèrement inférieures aux autres. Cependant, la taille du lot a un impact tangible sur le temps d'entraînement : un lot plus petit entraîne un temps d'entraînement plus long, tandis qu'un lot plus grand réduit ce temps.

Dans cette analyse, la taille de lot de 256 semble être l'option la plus équilibrée, offrant des résultats solides en termes de précision et de perte, tout en conservant un temps d'entraînement raisonnable.

4.2.1.5 Modèle final

Après une série exhaustive de tests et d'analyses, nous avons affiné une combinaison de paramètres pour obtenir le modèle optimal selon notre approche. L'architecture du modèle, inspirée d'un DenseNet, comporte quatre couches de convolution et des couches de transition. La taille de lot de 256 a été déterminée comme étant la plus efficace pour équilibrer la précision et la vitesse d'entraînement. Le modèle a été compilé avec l'optimiseur "adam" et une perte de "categorical_crossentropy", ce qui a été adapté à notre problème de classification multiclasse. Le modèle a été entraîné pour un total de 50 époques. Ci-dessous, vous trouverez le code de ce modèle sélectionné, reflétant les meilleures pratiques et les paramètres optimaux identifiés lors de nos expérimentations.

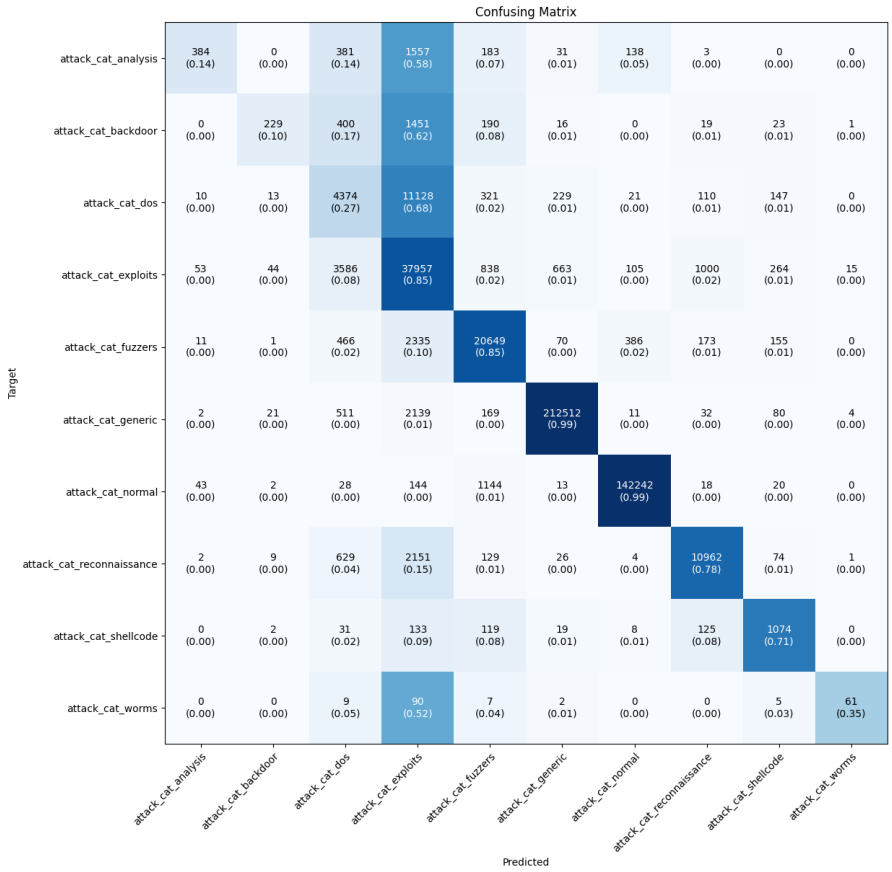
Figure 29 - Code du modèle final

```
def transition_layer(x, num_filters):
    x = Conv2D(num_filters, (1, 1), padding='same', activation='relu')(x)
    x = BatchNormalization()(x)
    return x

def best_model(input_img):
    # première couche de convolution
    x1 = Conv2D(32, (3, 3), padding='same', activation='relu')(input_img)
    x1 = BatchNormalization()(x1)
    # deuxième couche de convolution
    x2 = Conv2D(32, (3, 3), padding='same', activation='relu')(x1)
    x2 = BatchNormalization()(x2)
    concat2 = concatenate([x1, x2])
    concat2 = transition_layer(concat2, 32)  # couche de transition
    # troisième couche de convolution
    x3 = Conv2D(32, (3, 3), padding='same', activation='relu')(concat2)
    x3 = BatchNormalization()(x3)
    concat3 = concatenate([x1, x2, x3])
    concat3 = transition_layer(concat3, 32)  # couche de transition
    # quatrième couche de convolution
    x4 = Conv2D(32, (3, 3), padding='same', activation='relu')(concat3)
    x4 = BatchNormalization()(x4)
    concat4 = concatenate([x1, x2, x3, x4])
    concat4 = transition_layer(concat4, 32)  # couche de transition
    output = Flatten()(concat4)
    return Dense(nb_classes, activation='softmax')(output)

name = "Best model"
epochs = 50
batch_size = 256

# Définition de l'architecture du modèle
input_img = Input(shape=(8, 8, 1))
out = best_model(input_img)
model = Model(inputs=input_img, outputs=out)
model.compile(optimizer="adam", loss="categorical_crossentropy", metrics=["accuracy"])

# Entraînement du modèle
history = model.fit(x_train, y_train, validation_data=(x_test, y_test), verbose=1,
batch_size=batch_size, epochs=epochs)
```

Suite à l'entraînement de notre modèle, nous avons analysé les métriques suivantes :

- **Accuracy** :
  La précision globale du modèle est de 92,58%, indiquant la proportion de prédictions correctes.
- **Precision** :
  76,04% des prédictions positives étaient correctes, le reste étant des faux positifs.
- **Recall** :
  Le modèle a correctement identifié 60,35% des valeurs positives réelles.
- **F1 Score :**
  Le score F1 de 62,64%, en tant que moyenne harmonique de la précision et du rappel, indique un équilibre modéré entre ces critères de performance essentiels dans la classification.

Figure 30 - Matrice de confusion du modèle final



La matrice de confusion est un outil primordial pour évaluer la performance d'un modèle de classification. Dans cette matrice, l'axe vertical représente les valeurs réelles de notre jeu de données tandis que l'axe horizontal illustre les prédictions du modèle. Idéalement, nous aspirons à ce que la diagonale principale affiche un taux de précision de 100%.

Considérons, par exemple, la cellule située en haut à gauche de la matrice, où « Attack_cat_analysis » (valeur réelle) croise « Attack_cat_analysis » (valeur prédite). Cette intersection nous informe que le modèle a identifié avec justesse 384 instances comme étant du type « Attack_cat_analysis ». Cependant, il est notable que seulement 14% des attaques classifiées initialement comme "analysis" aient été correctement détectées par notre modèle. En examinant une autre cellule, celle associée à «

Attack_cat_analysis » (réel) et « attack_cat_exploits » (prédit), on observe que le modèle a malencontreusement classé 1557 instances comme étant des « exploits », alors qu'elles étaient réellement des « analysis », soit 58% des attaques de type « analysis » incorrectement catégorisées.

Il est manifeste que notre modèle est particulièrement performant pour certaines classes. Par exemple, il affiche une précision impressionnante de 99% pour les catégories "generic" et "normal". De plus, il démontre une capacité notable à prédire correctement les "fuzzers", les "reconnaissances" et les "shellcodes".

Cependant, certaines lacunes se démarquent. Le modèle peine notamment à classifier correctement les "exploits", avec une proportion non négligeable de faux positifs. De manière plus préoccupante, les performances du modèle chutent drastiquement pour les catégories "analyses" et "backdoors".

Cette évaluation met en relief les atouts et les points d'amélioration de notre modèle. Elle nous guide vers les domaines nécessitant des optimisations pour atteindre une performance exemplaire.

5 Conclusion

Au début de ce travail, nous avons examiné l'étendue et la complexité des différentes menaces que l'on peut rencontrer. Le nombre de menaces différentes est extrêmement élevé et leur classification dans des catégories est souvent compliquée, car les attaques peuvent appartenir à plusieurs catégories simultanément.

Nous avons également étudié les différents systèmes de détection d'infraction réseau, notamment les systèmes basés sur des signatures et les systèmes holistiques. Bien que les systèmes par signature soient extrêmement efficaces contre les menaces connues, ils sont vulnérables aux nouvelles attaques. En revanche, les systèmes holistiques ont tendance à générer de nombreux faux positifs.

Face à ces défis, nous avons décidé d'expérimenter une approche basée sur le Machine learning et le deep learning. En utilisant l'architecture des réseaux de neurones de convolution, réputée pour sa capacité à détecter des modèles dans les données, nous avons créé un modèle capable de classer les paquets en fonction des catégories d'attaque. Les résultats ont été prometteurs, avec une accuracy de 92% et une perte de 20%, bien que le modèle soit relativement léger.

Les principaux éléments de notre expérimentation comprennent :

- L'adoption d'une architecture DenseNet, qui s'est avérée efficace tout en ayant un nombre de paramètres bien inférieur aux autres architectures testées.
- Utilisation de quatre couches de convolution pour augmenter la précision.
- Incorporation de couches de transition adaptées aux petites images, réduisant considérablement le nombre de paramètres.
- Sélection d'un batch size de 256.
- Compilation du modèle avec l'optimiseur "adam" et la fonction de perte "categorical_crossentropy", ce qui a été crucial pour l'adaptation de notre problème de classification multiclasse.

Toutefois, malgré une efficacité manifeste, le modèle n'est pas utilisable en production. L'accuracy globale du modèle atteint 92.58%, mais les autres métriques révèlent des nuances importantes. La précision, par exemple, montre que seulement 76% des prédictions positives étaient correctes, et le recall indique que seulement 60% des instances positives ont été correctement identifiées. Certaines classes, comme "normal" et "generic", sont prédites avec une précision proche de 99%, tandis que les classes "backdoors" et "analysis" n'obtiennent que des scores de 10% et 14% respectivement.

Ces défis peuvent être partiellement attribués à l'hétérogénéité du jeu de données. L'utilisation d'un ensemble de données plus homogène ou l'incorporation de jeux de données supplémentaires, tels que TON_IoT et BoT_IoT, pourrait contribuer à affiner ces résultats. De plus, il serait judicieux d'explorer d'autres types de réseaux de neurones afin de rechercher des opportunités pour améliorer davantage les performances. L'examen de différentes fonctions d'activation et l'expérimentation avec des optimiseurs autres qu'Adam pourraient également apporter des améliorations. Enfin, la possibilité d'effectuer une classification binaire pour déterminer si un paquet constitue une attaque ou non aurait pu être explorée, car cela aurait peut-être conduit à de meilleurs résultats.

Pour les prochaines étapes de cette expérimentation, il pourrait être envisagé d'intégrer le modèle au sein d'un dispositif hardware compact, tel qu'un système de détection d'intrusions d’hôte (HIDS). Cela permettrait non seulement d'exploiter le modèle dans un environnement plus réaliste et contrôlé, mais aussi de procéder à une surveillance continue et en temps réel du réseau.

Bibliographie

1. Qu’est-ce que la cybersécurité ? | IBM, [sans date] [en ligne]. Disponible à l’adresse : https://www.ibm.com/fr-fr/topics/cybersecurity [consulté le 3 mai 2023].

2. Qu’est-ce que la cybersécurité ?, 2023*www.kaspersky.fr* [en ligne]. Disponible à l’adresse : https://www.kaspersky.fr/resource-center/definitions/what-is-cyber-security [consulté le 4 mai 2023].

3. CHENG, Eric C. K. et WANG, Tianchong, 2022. Institutional Strategies for Cybersecurity in Higher Education Institutions. Information. Vol. 13, no 4, p. 192. DOI 10.3390/info13040192.

4. TEAM, Pandora FMS, 2018. Creeper and Reaper, the First Virus and First Antivirus in History. Pandora FMS Monitoring Blog [en ligne]. 10 octobre 2018. Disponible à l’adresse : https://pandorafms.com/blog/creeper-and-reaper/ [consulté le 26 juin 2023].

5. Attaque par déni de service, 2023Wikipédia [en ligne]. Disponible à l’adresse : https://fr.wikipedia.org/w/index.php?title=Attaque_par_d%C3%A9ni_de_service&oldid=203696311 [consulté le 8 mai 2023]. Page Version ID: 203696311

6. Qu’est-ce qu’une attaque DDoS ?, [sans date]Cloudflare [en ligne]. Disponible à l’adresse : https://www.cloudflare.com/fr-fr/learning/ddos/what-is-a-ddos-attack/ [consulté le 8 mai 2023].

7. Attaque DDoS Smurf, [sans date]Cloudflare [en ligne]. Disponible à l’adresse : https://www.cloudflare.com/fr-fr/learning/ddos/smurf-ddos-attack/ [consulté le 19 juillet 2023].

8. DoS vs. DDoS Attack: What Is the Difference? - Radware, [sans date] [en ligne]. Disponible à l’adresse : https://www.radware.com/cyberpedia/ddos-attacks/dos-vs-ddos-attack-what-is-the-difference/ [consulté le 20 juillet 2023].

9. Fuzzing, 2020Wikipédia [en ligne]. Disponible à l’adresse : https://fr.wikipedia.org/w/index.php?title=Fuzzing&oldid=174149795 [consulté le 10 mai 2023]. Page Version ID: 174149795

10. Fuzzing | OWASP Foundation, [sans date] [en ligne]. Disponible à l’adresse : https://owasp.org/www-community/Fuzzing [consulté le 14 mai 2023].

11. What is Fuzzing (Fuzz Testing)? | Tools, Attacks & Security | Imperva, [sans date]Learning Center [en ligne]. Disponible à l’adresse : https://www.imperva.com/learn/application-security/fuzzing-fuzz-testing/ [consulté le 15 mai 2023].

12. SIROS, Ilja, 2022. Wireless Network Protocol Fuzzing. COSIC [en ligne]. 17 mars 2022. Disponible à l’adresse : https://www.esat.kuleuven.be/cosic/blog/wireless-network-protocol-fuzzing/ [consulté le 15 mai 2023].

13. Port scanner, 2023Wikipedia [en ligne]. Disponible à l’adresse : https://en.wikipedia.org/w/index.php?title=Port_scanner&oldid=1141578388 [consulté le 27 juin 2023]. Page Version ID: 1141578388

14. BAELDUNG, 2022. Port Scanning Explained | Baeldung on Computer Science. [en ligne]. 12 septembre 2022. Disponible à l'adresse : https://www.baeldung.com/cs/port-scanning [consulté le 27 juin 2023].

15. What Is Port Scanning? - Datto Networking, [sans date] [en ligne]. Disponible à l'adresse : https://www.datto.com/blog/what-is-port-scanning?utm_medium=opengraph&utm_source=225 [consulté le 26 juin 2023].

16. Backdoor (computing), 2023Wikipedia [en ligne]. Disponible à l'adresse : https://en.wikipedia.org/w/index.php?title=Backdoor_(computing)&oldid=1152837946 [consulté le 28 juin 2023]. Page Version ID: 1152837946

17. What is a Backdoor Attack, [sans date] [en ligne]. Disponible à l'adresse : https://www.tutorialspoint.com/what-is-a-backdoor-attack [consulté le 27 juin 2023].

18. exploiter - Définition, [sans date] [en ligne]. Disponible à l'adresse : https://www.trendmicro.com/vinfo/us/security/definition/exploit [consulté le 29 juin 2023].

19. Exploits, [sans date]Hackers Online Club (HOC) [en ligne]. Disponible à l'adresse : https://hackersonlineclub.com/exploits/ [consulté le 30 juin 2023].

20. ARTYKOV, David, 2021. Information gathering using DNS enumeration techniques and tools. *Geek Culture* [en ligne]. 28 mai 2021. Disponible à l'adresse : https://medium.com/geekculture/dns-enumeration-3dc90ca1f670 [consulté le 19 août 2023].

21. SMTP Enumeration, 2022*GeeksforGeeks* [en ligne]. Disponible à l'adresse : https://www.geeksforgeeks.org/smtp-enumeration/ [consulté le 19 août 2023].

22. SunRPC Portmap Program Enumerator - Metasploit, [sans date]*InfosecMatter* [en ligne]. Disponible à l'adresse : https://www.infosecmatter.com/metasploit-module-library/?mm=auxiliary/scanner/misc/sunrpc_portmapper [consulté le 19 août 2023].

23. Shellcode, 2023Wikipedia [en ligne]. Disponible à l'adresse : https://en.wikipedia.org/w/index.php?title=Shellcode&oldid=1171108927#cite_note-1 [consulté le 19 août 2023]. Page Version ID: 1171108927

24. FOSTER, James C., 2005. Sockets, Shellcode, Porting, and Coding: Reverse Engineering Exploits and Tool Coding for Security Professionals. Elsevier. ISBN 978-0-08-048972-8. Google-Books-ID: ZNI5dvBSfZoC

25. Virus informatique, 2022Wikipédia [en ligne]. Disponible à l'adresse : https://fr.wikipedia.org/w/index.php?title=Virus_informatique&oldid=197618454 [consulté le 19 août 2023]. Page Version ID: 197618454

26. Computer worm, 2023Wikipedia [en ligne]. Disponible à l'adresse : https://en.wikipedia.org/w/index.php?title=Computer_worm&oldid=1170845433 [consulté le 19 août 2023]. Page Version ID: 1170845433

27. MISHRA, Bimal Kumar et PRAJAPATI, Apeksha, 2014. Cyber Warfare : Worms' Transmission Model. International Journal of Advanced Science and Technology. Vol. 63, pp. 83-94. DOI 10.14257/ijast.2014.63.08.

28. What is a computer worm, and how does it work?, [sans date] [en ligne]. Disponible à l'adresse : https://us.norton.com/blog/malware/what-is-a-computer-worm [consulté le 24 juillet 2023].

29. What is SQL Injection? Tutorial & Examples | Web Security Academy, [sans date] [en ligne]. Disponible à l'adresse : https://portswigger.net/web-security/sql-injection [consulté le 19 août 2023].

30. BHARDWAJ, Rashmi, 2020. IDS vs IPS vs Firewall - Know the Difference - IP With Ease. [en ligne]. 10 septembre 2020. Disponible à l'adresse : https://ipwithease.com/firewall-vs-ips-vs-ids/ [consulté le 19 août 2023].

31. Intrusion Detection System (IDS), 2019GeeksforGeeks [en ligne]. Disponible à l'adresse : https://www.geeksforgeeks.org/intrusion-detection-system-ids/ [consulté le 19 août 2023].

32. What is an Intrusion Detection System (IDS)?, [sans date]Check Point Software [en ligne]. Disponible à l'adresse : https://www.checkpoint.com/cyber-hub/network-security/what-is-an-intrusion-detection-system-ids/ [consulté le 19 août 2023].

33. REZEK, Michael, 2020. What is the difference between signature-based and behavior-based intrusion detection systems? Accedian [en ligne]. 9 décembre 2020. Disponible à l'adresse : https://accedian.com/blog/what-is-the-difference-between-signature-based-and-behavior-based-ids/ [consulté le 19 août 2023].

34. BRINDHA, Dr S et ABIRAMI, P, 2020. Heuristic Approach to Intrusion Detection System. . Vol. 07, no 03.

35. Research and Implementation on Snort-Based Hybrid Intrusion Detection System - Dissertation, [sans date] [en ligne]. Disponible à l'adresse : https://m.dissertationtopic.net/doc/283988 [consulté le 19 août 2023].

36. Apprentissage automatique, 2023Wikipédia [en ligne]. Disponible à l'adresse : https://fr.wikipedia.org/w/index.php?title=Apprentissage_automatique&oldid=206902810 [consulté le 17 août 2023]. Page Version ID: 206902810

37. Machine Learning : Définition, fonctionnement, utilisations, 2020Formation Data Science | DataScientest.com [en ligne]. Disponible à l'adresse : https://datascientest.com/machine-learning-tout-savoir [consulté le 19 août 2023].

38. Weak supervision, 2023Wikipedia [en ligne]. Disponible à l'adresse : https://en.wikipedia.org/w/index.php?title=Weak_supervision&oldid=1167373240 [consulté le 19 août 2023]. Page Version ID: 1167373240

39. Apprentissage supervisé et non supervisé : les différencier et les combiner, [sans date]LeMagIT [en ligne]. Disponible à l'adresse : https://www.lemagit.fr/conseil/Apprentissage-supervise-et-non-supervise-les-differencier-et-les-combiner [consulté le 19 août 2023].

40. MISHRA, Chandrahas et GUPTA, Dharmendra Lal, 2016. Deep Machine Learning and Neural Networks: An Overview. International Journal of Hybrid Information Technology. Vol. 9, no 11, pp. 401-414. DOI 10.14257/ijhit.2016.9.11.34.

41. Vers une détection d'intrusion dynamique et continue en utilisant les réseaux de neurones | Connect - Editions Diamond, [sans date] [en ligne]. Disponible à l'adresse : https://connect.ed-diamond.com/MISC/mischs-018/vers-une-detection-d-intrusion-dynamique-et-continue-en-utilisant-les-reseaux-de-neurones [consulté le 26 juin 2023].

42. Artificial neural network, 2023Wikipedia [en ligne]. Disponible à l'adresse : https://en.wikipedia.org/w/index.php?title=Artificial_neural_network&oldid=1170800588 [consulté le 19 août 2023]. Page Version ID: 1170800588

43. CHOWDHURY, Md Moin Uddin et al., 2017. A few-shot deep learning approach for improved intrusion detection. In : 2017 IEEE 8th Annual Ubiquitous Computing, Electronics and Mobile Communication Conference (UEMCON), pp. 456-462. New York, NY, USA : IEEE. octobre 2017. ISBN 978-1-5386-1104-3. DOI 10.1109/UEMCON.2017.8249084.

44. Qu'est ce qu'un réseau de neurones convolutif (ou CNN) ?, [sans date]OpenClassrooms [en ligne]. Disponible à l'adresse : https://openclassrooms.com/fr/courses/4470531-classez-et-segmentez-des-donnees-visuelles/5082166-quest-ce-quun-reseau-de-neurones-convolutif-ou-cnn [consulté le 19 août 2023].

45. Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow, 3rd Edition [Book], [sans date] [en ligne]. Disponible à l'adresse : https://www.oreilly.com/library/view/hands-on-machine-learning/9781098125967/ [consulté le 17 août 2023].

46. Réseaux convolutifs (CNN) : comment ça marche ?, [sans date]Formation Tech et Data en ligne | Blent.ai [en ligne]. Disponible à l'adresse : https://blent.ai/blog/a/cnn-comment-ca-marche [consulté le 19 août 2023].

47. CNN | Introduction to Pooling Layer, 2019GeeksforGeeks [en ligne]. Disponible à l'adresse : https://www.geeksforgeeks.org/cnn-introduction-to-pooling-layer/ [consulté le 19 août 2023].

48. KELDENICH, Tom, 2021. Fonction d'activation, comment ça marche ? - Une explication simple. Inside Machine Learning [en ligne]. 8 février 2021. Disponible à l'adresse : https://inside-machinelearning.com/fonction-dactivation-comment-ca-marche-une-explication-simple/ [consulté le 19 août 2023].

49. Dense In Deep Learning - Coding Ninjas, [sans date] [en ligne]. Disponible à l'adresse : https://www.codingninjas.com/studio/library/dense-in-deep-learning [consulté le 19 août 2023].

50. KELDENICH, Tom, 2021. 7 types of Layers you need to know in Deep Learning and how to use them. Inside Machine Learning [en ligne]. 27 octobre 2021. Disponible à l'adresse : https://inside-machinelearning.com/en/7-types-of-layers-you-need-to-know-in-deep-learning-and-how-to-use-them/ [consulté le 19 août 2023].

51. BAELDUNG, 2020. How ReLU and Dropout Layers Work in CNNs | Baeldung on Computer Science. [en ligne]. 30 mai 2020. Disponible à l'adresse : https://www.baeldung.com/cs/ml-relu-dropout-layers [consulté le 19 août 2023].

52. API Reference, [sans date]scikit-learn [en ligne]. Disponible à l'adresse : https://scikit-learn/stable/modules/classes.html [consulté le 19 août 2023].

53. TEAM, Dst, 2020. Algorithme de descente de gradient - Machine Learning. Formation Data Science | DataScientest.com [en ligne]. 20 juillet 2020. Disponible à l'adresse : https://datascientest.com/descente-de-gradient [consulté le 15 août 2023].

54. LEARNIA, 2019. DESCENTE DE GRADIENT (GRADIENT DESCENT) - ML#4 [en ligne]. 11 juillet 2019. Disponible à l'adresse : https://www.youtube.com/watch?v=rcl_YRyoLIY [consulté le 15 août 2023].

55. ML | Stochastic Gradient Descent (SGD), 2019GeeksforGeeks [en ligne]. Disponible à l'adresse : https://www.geeksforgeeks.org/ml-stochastic-gradient-descent-sgd/ [consulté le 19 août 2023].

56. BROWNLEE, Jason, 2017. Gentle Introduction to the Adam Optimization Algorithm for Deep Learning. MachineLearningMastery.com [en ligne]. 2 juillet 2017. Disponible à l'adresse : https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/ [consulté le 13 août 2023].

57. ML | ADAM (Adaptive Moment Estimation) Optimization, 2020GeeksforGeeks [en ligne]. Disponible à l'adresse : https://www.geeksforgeeks.org/adam-adaptive-moment-estimation-optimization-ml/ [consulté le 19 août 2023].

58. MOUSTAFA, Nour, CREECH, Gideon et SLAY, Jill, 2017. Big Data Analytics for Intrusion Detection System: Statistical Decision-Making Using Finite Dirichlet Mixture Models. In : PALOMARES CARRASCOSA, Iván, KALUTARAGE, Harsha Kumara et HUANG, Yan (éd.), Data Analytics and Decision Support for Cybersecurity: Trends, Methodologies and Applications, pp. 127-156. Cham : Springer International Publishing. Data Analytics. ISBN 978-3-319-59439-2. DOI 10.1007/978-3-319-59439-2_5.

59. MOUSTAFA, Nour et SLAY, Jill, 2015. UNSW-NB15: a comprehensive data set for network intrusion detection systems (UNSW-NB15 network data set). In : 2015 Military Communications and Information Systems Conference (MilCIS), pp. 1-6. novembre 2015. DOI 10.1109/MilCIS.2015.7348942.

60. MOUSTAFA, Nour et SLAY, Jill, 2016. The evaluation of Network Anomaly Detection Systems: Statistical analysis of the UNSW-NB15 data set and the comparison with the KDD99 data set. Information Security Journal: A Global Perspective. Vol. 25, no 1-3, pp. 18-31. DOI 10.1080/19393555.2015.1125974.

61. MOUSTAFA, Nour, SLAY, Jill et CREECH, Gideon, 2019. Novel Geometric Area Analysis Technique for Anomaly Detection Using Trapezoidal Area Estimation on Large-Scale Networks. IEEE Transactions on Big Data. Vol. 5, no 4, pp. 481-494. DOI 10.1109/TBDATA.2017.2715166.

62. CHARLESMURE, 2022. cassiope-NIDS [logiciel] [en ligne]. 14 août 2022. [consulté le 19 août 2023]. Disponible à l'adresse : https://github.com/CharlesMure/cassiope-NIDS [consulté le 19 août 2023].

63. FARIDBG, 2023. CNN_UNSW-NB15 [logiciel] [en ligne]. 11 août 2023. [consulté le 19 août 2023]. Disponible à l'adresse : https://github.com/Faridbg/CNN_UNSW-NB15 [consulté le 19 août 2023].

6 Annexe

6.1 Annexe 1 : Les différents types de scan de ports

1. **TCP SYN Scan :**
   C’est une des techniques de balayage de ports les plus répandues. Sa rapidité et sa discrétion, dues au fait qu'elle n'aboutit pas à une connexion TCP complète, en font une méthode privilégiée.

   Le principe de cette technique repose sur l'utilisation des paquets SYN, spécifiques au protocole TCP.

   Voici comment se déroule une connexion TCP normale, connue sous le nom de "processus en trois étapes" :
   1. L'initiateur de la connexion envoie un paquet SYN au système cible.
   2. Si le port cible est ouvert, le système récepteur répond avec un paquet SYN/ACK, signalant sa disponibilité pour établir une connexion.
   3. Enfin, l'initiateur envoie un paquet ACK pour finaliser la connexion.

   Lors d'un TCP SYN Scan, la troisième étape n'est pas exécutée. Au lieu de cela, le scanner se concentre sur les réponses de la deuxième étape. Si un paquet SYN/ACK est reçu, cela indique que le port est ouvert et prêt à recevoir des données. Un paquet RST signifie que le port est fermé. Enfin, si aucune réponse n'est reçue malgré plusieurs tentatives, le port est considéré comme filtré.

   Le TCP SYN Scan se révèle donc une méthode puissante pour identifier rapidement les ports ouverts d'un système, offrant un avantage stratégique pour évaluer la sécurité du réseau [14].

2. **TCP Connect Scan :**
   Cette technique est une autre méthode de balayage de ports. Contrairement au TCP SYN Scan qui n'établit pas une connexion TCP complète, le TCP Connect Scan, quant à lui, instruit le système d'exploitation à établir une réelle connexion TCP avec le système cible. C'est une alternative privilégiée lorsque l'utilisateur ne dispose pas des privilèges requis pour l'envoi de paquets bruts (raw packets), nécessaires dans le cas du TCP SYN Scan.

   Voici comment se déroule le TCP Connect Scan :
   1. L'outil de balayage débute en envoyant une demande de connexion, sous la forme d'un paquet SYN, vers un port spécifique de la machine cible.
   2. Si le port est accessible, la machine cible répond avec un paquet SYN/ACK.
   3. En réponse, l'outil de balayage envoie un paquet ACK pour finaliser l'établissement de la connexion TCP.

   Comparé au TCP SYN Scan, le TCP Connect Scan est plus long et nécessite plus de paquets pour recueillir la même quantité d'informations. Cette différence s'explique par l'établissement d'une véritable connexion TCP.

   Cependant, cette méthode est moins discrète. Du fait de la connexion complète établie, le système cible est plus enclin à enregistrer l'interaction dans ses journaux d'événements, rendant ainsi cette tentative de balayage plus facile à détecter par les administrateurs réseau [14].

3. **UDP scanning :**
   Cette méthode consiste à envoyer un paquet à un port spécifique. Si le port est fermé, le système retournera un message ICMP "Port inaccessible". Par conséquent, l'absence de réponse suggère qu'un port est ouvert. Cependant, si un port est verrouillé par un pare-feu, cette méthode pourrait faussement rapporter que le port est ouvert.

   Une variante du balayage UDP implique l'envoi de paquets UDP spécifiques à une application donnée. Par exemple, l'envoi d'une requête DNS au port 53 engendrera une réponse si un serveur DNS est en fonction. Cette technique est nettement plus fiable pour détecter les ports ouverts. Néanmoins, elle est limitée au balayage de ports pour lesquels un paquet adapté à une application précise est disponible [14].

4. **ACK scanning :**
   Cette technique scan de ports ne permet pas de déterminer directement si un port est ouvert ou fermé. Cependant, elle permet d'identifier si un port est filtré ou non, ce qui aide à déceler la présence d'un pare-feu et à comprendre les règles de filtrage associées.

   Voici en détail le fonctionnement du balayage ACK :
   1. Le scanner envoie un paquet portant le drapeau ACK vers un port spécifique sur la machine cible.
   2. Si le port est filtré par un pare-feu, aucun paquet de réponses ne sera renvoyé. En revanche, si le port est non filtré, la machine cible renverra un paquet RST (Reset).

   Cette forme de balayage est particulièrement utile pour analyser le comportement d'un pare-feu vis-à-vis des connexions déjà établies [13].

6.2 Annexe 2 : Différents types d'apprentissage du ML

- **En apprentissage supervisé :**
  Dans cette approche d'apprentissage, le jeu de données contient des entrées ainsi que leurs étiquettes de sortie correspondantes. Cette méthode nécessite généralement moins de données pour l'entraînement que les autres approches, ce qui simplifie le processus d'apprentissage. Cependant, l'obtention de données correctement étiquetées peut se révéler particulièrement coûteuse. [37]

- **En apprentissage non supervisé :**
  Dans cette approche, l'apprentissage est effectué uniquement à partir des données d'entrée, qui ne sont pas étiquetées. L'objectif de l'algorithme est de créer des représentations utiles pour les prédictions futures. L'algorithme utilisera des techniques pour identifier les caractéristiques pertinentes nécessaires à l'étiquetage, au tri et à la classification des données. De plus, cette méthode peut permettre de découvrir des modèles et des relations que les êtres humains n'auraient pas été en mesure de détecter. Bien que son application puisse s'avérer complexe, rendant son utilisation moins courante, elle est particulièrement prisée dans le domaine de la cybersécurité. [37]

- **Apprentissage Semi-supervisé ?**
  L'émergence des Modèles de Langage à Grande Échelle (LLM) a considérablement favorisé l'essor d'une certaine forme d'apprentissage, caractérisée par la combinaison d'apprentissage supervisé et non supervisé. Cette méthode est particulièrement pertinente lorsqu'il y a une disproportion entre le volume de données non étiquetées, qui est généralement plus important, et la petite quantité de données étiquetées. [38]

  Dans cette démarche, les algorithmes d'apprentissage non supervisé jouent un rôle crucial. Ils sont utilisés pour générer automatiquement des étiquettes qui peuvent être ensuite utilisées par les algorithmes d'apprentissage supervisé. Pour illustrer, prenons l'exemple des images. Certaines images sont d'abord étiquetées manuellement. Ensuite, l'apprentissage non supervisé est utilisé pour prédire les étiquettes des autres images. Toutes ces images et leurs étiquettes respectives sont ensuite introduites dans un algorithme d'apprentissage supervisé. [39]

6.3 Annexe 3 : La descente de gradient

Descente de gradient : (en annexe)
La descente de gradient est une méthode d'optimisation couramment employée pour réduire l'erreur d'un modèle de deep learning. En essence, cette technique vise à affiner les prédictions d'un modèle en minimisant la différence entre ses prédictions et les données réelles. [53]

Pour illustrer cette technique, imaginez-vous debout au sommet d'une montagne vallonnée, cherchant le chemin le plus rapide pour descendre dans la vallée. À chaque étape, vous utilisez une boussole qui vous indique la direction de la pente la plus raide. Vous suivez cette direction, avançant pas à pas, jusqu'à ce que vous atteigniez un point bas où le terrain est plat.

Dans cette analogie :

- Votre position actuelle sur la montagne correspond à x_t
- La boussole, qui vous indique la direction de la descente la plus rapide, est l'équivalent du gradient de la fonction à ce point.

Le paramètre η, appelé taux d'apprentissage, détermine la taille de chaque pas que vous faites. Si η est trop grand, vous ferez de grands pas, ce qui pourrait vous permettre de descendre plus rapidement, mais avec le risque de dépasser le fond de la vallée ou de manquer des minima locaux. À l'inverse, si η est trop petit, vous progresserez lentement avec de nombreux petits pas, ce qui pourrait rallonger considérablement le temps nécessaire pour atteindre le minimum. [54]

En somme, la descente de gradient est une méthode itérative qui ajuste progressivement la position d'un point dans l'espoir de trouver un minimum local ou global de la fonction de coût. [53]

6.4 Annexe 4 : La descente de gradient stochastique (SGD)

La descente de gradient stochastique (SGD) est une version optimisée de la descente de gradient, spécifiquement conçue pour traiter les défis associés aux grands ensembles de données dans le contexte du deep learning. La descente de gradient classique, bien qu'efficace, peut être coûteuse en termes de temps et de ressources lorsque l'on travaille avec de vastes volumes de données.

Contrairement à la descente de gradient traditionnelle qui utilise l'ensemble de données entier à chaque itération pour calculer le gradient, la SGD sélectionne aléatoirement un échantillon ou un petit lot d'échantillons lors de chaque étape.

Le principal avantage de la SGD est son efficacité en matière de calcul. En réduisant considérablement la quantité de données traitées à chaque étape, la SGD accélère grandement le processus d'optimisation, rendant ainsi le traitement de grands ensembles de données plus viable. Cependant, cette efficacité vient avec un compromis : étant donné la nature aléatoire de la sélection des échantillons, la SGD peut nécessiter davantage d'itérations pour converger vers le minimum optimal de la fonction de coût.

En résumé, bien que la descente de gradient stochastique puisse introduire une certaine variabilité dans le processus d'optimisation, elle offre une solution plus rapide et plus adaptable pour traiter les grands ensembles de données couramment rencontrés en deep learning. [55]

6.5 Annexe 5 : Explication du dataset UNSW-NB15

Notre réalisation utilise le jeu de données UNSW-NB15, qui a été créé par l'outil IXIA PerfectStorm dans le Cyber Range Lab du Australian Centre for Cyber Security (ACCS). Avant la création de ce jeu de donnée, les jeux de données existants étaient beaucoup trop vieux et ne représentaient plus des attaques contemporaines. C'est pourquoi le dataset UNSW-NB15 est constitué d'une grande quantité d'activités normales ainsi que des attaques contemporaines. L'outil tcpdump a été utilisé pour capturer toutes les données de paquets bruts, le jeu de données contient plus de 100 Go de données et compte 2'540'044 enregistrements. Chaque enregistrement contient plus de 49 caractéristiques, y compris l'étiquette de la classe d'attaque. Ces caractéristiques ont été créées avec les outils Argus, Bro-IDS et douze algorithmes. Le jeu de données contient 9 catégories d'attaques et la catégorie normale.

Figure 31 - Tableau des Différents Types d'Attaques dans le Jeu de Données UNSW-NB15

Type d'attaques	Nombre d'enregistrement
Normal	2'218'764
Generic	215'481
Exploits	44'525
Fuzzers	24'246
DoS	16'353
Reconnaissance	13'987
Analysis	2'677
Backdoors	2'329
Shellcode	1'511
Worms	174

Le jeu de données a été divisé en quatre fichiers CSV, chacun contenant à la fois des attaques et des activités normales.

Figure 32 - Différents fichiers du dataset UNSW-NB15

Nom du fichier	Taille du fichier	Nombre d'enregistrements	Nombre de features
UNSWNB15_1.csv	161.2 MB	700'000	49
UNSWNB15_2.csv	157.6 MB	700'000	49
UNSWNB15_3.csv	147.4 MB	700'000	49
UNSWNB15_4.csv	91.3 MB	440'044	49

Ce dataset contient 49 features en comptant la variable cible. Chaque feature sont de 3 différents types.

- Catégorique : variables qualitatives prenant un nombre fini de catégories
- Binaire : variables qui peuvent prendre 0 ou 1
- Numérique : variables quantitatives prenant des valeurs numériques

Les différents attributs avec leurs descriptions sont disponibles en **Annexe 6 : Description détaillée des colonnes du jeu de données.**

6.6 Annexe 6 : Description détaillée des colonnes du jeu de données.

Figure 33 - Tableau de la description du jeu de données UNSW-NB15

No.	Nom	Type de donnée	Description
1	srcip	Numérique	Adresse IP source
2	sport	Numérique	Port source
3	dstip	Numérique	Adresse IP destination
4	dsport	Numérique	Port destination
5	proto	Catégorique	Protocole de transaction
6	state	Catégorique	Indication de l'état du protocole (ACC, CLO, CON, ECO, ECR, FIN, INT, MAS, PAR, REQ, RST, TST, TXD, URH, URN, et (-) (si l'état n'est pas utilisé).
7	Dur	Numérique	Enregistrement de la durée totale
8	Sbytes	Numérique	Octets de transaction entre la source et la destination
9	Dbytes	Numérique	Octets de transaction entre la destination et la source
10	Sttl	Numérique	Source – destination - durée de vie - valeur
11	dttl	Numérique	Destination - source - durée de vie - valeur
12	sloss	Numérique	Paquets sources retransmis ou abandonnés
13	dloss	Numérique	Paquets de destinations retransmis ou abandonnés
14	service	Catégorique	http, ftp, smtp, ssh, dns, ftp-data ,irc et (-) si service peu utilisé
15	Sload	Numérique	Bits de source par seconde
16	Dload	Numérique	Destination bits par seconde
17	Spkts	Numérique	Nombre de paquets de la source à la destination
18	Dpkts	Numérique	Nombre de paquets de la destination à la source
19	Swin	Numérique	Valeur d'annonce de la fenêtre TCP source
20	dwin	Numérique	Valeur d'annonce de la fenêtre TCP de destination
21	stcpb	Numérique	Numéro de séquence de la base TCP de la source
22	dtcpb	Numérique	Numéro de séquence de la base TCP de destination
23	smeansz	Numérique	La taille moyenne des paquets envoyés par la source (src) du flux réseau.
24	dmeansz	Numérique	La taille moyenne des paquets envoyés par la destination (dst) du flux réseau.
25	trans_depth	Numérique	Représente la profondeur du pipeline dans la connexion de la transaction requête/réponse http
26	res_bdy_len	Numérique	Taille réelle du contenu non compressé des données transférées par le service http du serveur.
27	Sjit	Numérique	Gigue de la source (mSec)
28	Djit	Numérique	Gigue de destination (mSec)

29	Stime	Numérique	Enregistrer l'heure de début
30	Ltime	Numérique	Record de la dernière fois
31	Sintpkt	Numérique	Temps d'arrivée de l'interpaquet de la source (mSec)
32	Dintpkt	Numérique	Temps d'arrivée de l'interpaquet de destination (mSec)
33	Tcprtt	Numérique	Temps d'établissement d'une connexion TCP, somme de synack et ackdat
34	synack	Numérique	Temps d'établissement de la connexion TCP, temps écoulé entre les paquets SYN et SYN_ACK.
35	ackdat	Numérique	Temps d'établissement de la connexion TCP, temps écoulé entre les paquets SYN_ACK et ACK.
36	is_sm_ips_ports	Binaire	Si les adresses IP de la source (1) et de la destination (3) sont égales et si les numéros de port (2)(4) sont égaux, cette variable prend la valeur 1, sinon la valeur 0.
37	ct_state_ttl	Numérique	N° pour chaque état (6) en fonction d'une gamme spécifique de valeurs pour la durée de vie de la source/destination (10) (11).
38	ct_flw_http_mthd	Numérique	Nombre de flux ayant des méthodes telles que Get et Post dans le service http.
39	is_ftp_login	Binaire	Si la session ftp est accessible par l'utilisateur et le mot de passe, alors 1 sinon 0.
40	ct_ftp_cmd	Numérique	Nombre de flux ayant une commande dans la session ftp.
41	ct_srv_src	Numérique	Nombre de connexions contenant le même service (14) et la même adresse source (1) sur 100 connexions selon la dernière heure (26).
42	ct_srv_dst	Numérique	Nombre de connexions contenant le même service (14) et la même adresse de destination (3) sur 100 connexions selon la dernière heure (26).
43	ct_dst_ltm	Numérique	Nombre de connexions de la même adresse de destination (3) sur 100 connexions selon la dernière heure (26).
44	ct_src_ ltm	Numérique	Nombre de connexions de la même adresse source (1) sur 100 connexions selon la dernière heure (26).
45	ct_src_dport_ltm	Numérique	Nombre de connexions de la même adresse source (1) et du même port de destination (4) sur 100 connexions selon la dernière heure (26).
46	ct_dst_sport_ltm	Numérique	Nombre de connexions de la même adresse de destination (3) et du même port source (2) sur 100 connexions selon la dernière heure (26).
47	ct_dst_src_ltm	Numérique	Nombre de connexions de la même adresse de source (1) et de destination (3) sur 100 connexions selon la dernière heure (26).

48	attack_cat	Catégorique	Le nom de chaque catégorie d'attaque. Dans cet ensemble de données, neuf catégories, à savoir Fuzzers, Analysis, Backdoors, DoS Exploits, Generic, Reconnaissance, Shellcode et Worms, ont été retenues.
49	Label	Binaire	0 pour les enregistrements normaux et 1 pour les enregistrements d'attaque

(Repris et traduit du dataset UNSW-NB15)