RÉPUBLIQUE ALGÉRIENNE DÉMOCRATIQUE ET POPULAIRE

Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

UNIVERSITÉ SAAD DAHLAB BLIDA 1

FACULTÉ DES SCIENCES

Département d'Informatique

MÉMOIRE DE PROJET DE FIN D'ETUDE

En vue de l'obtention du diplôme de Master en Informatique

OPTION : INGÉNIERIE LOGICIEL (IL)

Thème

Application des Méta-heuristiques pour l'Optimisation du Processus de Machine Learning

Réalisé par :

SALMI Chahinaz

MAATSEKI Selma

Encadré par :

Mme. AROUSSI Sana

Soutenu le 03 Juillet 2025 devant le jury :

Mr. OULD KHAOUA (Président). Mme. Hireche (Examinatrice)

REMERCIEMENTS

Avant toute chose, nous rendons grâce à ALLAH, Le Tout-Puissant, pour nous avoir accordé la force, la patience et la persévérance nécessaires pour mener à bien ce projet. Sans Sa volonté, rien n'aurait été possible.

Nous tenons à exprimer notre profonde gratitude à notre encadrante, Madame Sana AROUSSI, pour son encadrement rigoureux, sa disponibilité constante et ses conseils précieux tout au long de ce modeste travail. Sa bienveillance, son exigence intellectuelle et son soutien ont contribué largement à la qualité de ce mémoire.

Nous tenons également à remercier les membres du jury qui nous ont fait l'honneur de bien vouloir évaluer notre travail, nous vous remercions d'avance pour vos remarques et conseils pour l'enrichissement de ce mémoire.

À tous ceux qui ont contribué, de près ou de loin, à l'aboutissement de ce travail.

DÉDICACE

À ma maman bien-aimée, Mon frère Islam, vous êtes les piliers de ma vie. Votre amour inconditionnel, vos prières incessantes et votre confiance en moi m'ont portée et soutenue jusqu'à la réalisation de ce travail. Rien de tout cela n'aurait été possible sans vous.

À toute ma famille et tous ceux qui m'ont aidé de près ou de loin.

À tous mes enseignants, mes camarades, et à ceux qui ont partagé cette aventure, merci pour les échanges enrichissants, le soutien mutuel et les souvenirs inoubliables.

Avec toute ma gratitude et mon respect,

SALMI Chahinaz

DÉDICACE

À mes parents, mes héros de toujours, votre amour, vos sacrifices et votre foi en moi sont les plus précieux cadeaux que la vie m'ait offerts. Ce mémoire est aussi le vôtre, fruit de votre soutien sans faille et de votre présence rassurante à chaque instant.

À mes frères, sœurs et toute ma famille pour votre tendresse, votre écoute et votre soutien moral, même dans le silence.

À mes enseignants de l'Université Saad Dahlab de Blida Pour avoir partagé leur savoir avec passion et professionnalisme. Grâce à vous, j'ai évolué dans un cadre propice à l'apprentissage et à l'épanouissement.

À mes ami(e)s et camarades, Pour les moments d'entraide, de complicité et de solidarité qui ont rythmé cette belle aventure universitaire. Vous avez été une force précieuse.

À tous les chercheurs et penseurs de notre époque, vos travaux m'ont guidée, inspirée et ont enrichi la qualité de ce mémoire. Merci pour votre passion et votre quête du savoir

Avec un amour éternel

MAATSEKI Selma

RÉSUMÉ

Ce mémoire s'inscrit dans le cadre de l'optimisation des performances des modèles d'apprentissage automatique, en s'appuyant sur l'utilisation de l'algorithme Génétiques (GA). L'objectif principal est de proposer une approche d'optimisation conjointe des hyper paramètres et de sélection automatique de quatre modèles de l'apprentissage profond (DL) : MLP, DNN, CNN et LSTM. Le processus d'optimisation mis en place repose sur une modélisation à la fois mono-objectif (axée sur le Recall) et multi-objectif (axée sur le F1-score), afin de guider la recherche vers des modèles à haute capacité de généralisation. L'algorithme génétique est chargé d'explorer l'espace des hyper-paramètres, tout en identifiant dynamiquement l'architecture de réseau la plus adaptée aux données traitées.

Les expérimentations réalisées ont permis de comparer cette approche aux méthodes classiques d'optimisation comme la recherche exhaustive (Grid Search), la recherche aléatoire (Random Search) et l'optimisation Bayésienne mettant en évidence une amélioration notable en termes de précision, de temps d'exécution et de robustesse des modèles générés.

Mots clés : Méta heuristiques, Algorithme génétique, Deep learning, MLP, DNN, CNN, LSTM, Grid Search, Random Search, Optimisation Bayésienne.

ABSTRACT

This thesis focuses on optimizing the performance of machine learning models through the use of Genetic Algorithms (GA). The main objective is to propose a joint optimization approach for hyper parameters and automatic selection among four deep learning (DL) models: MLP, DNN, CNN, and LSTM. The implemented optimization process relies on both a single-objective formulation (focused on Recall) and a multi-objective formulation (incorporating metrics such as Precision, F1-score, etc.), in order to guide the search toward models with high generalization capability. The genetic algorithm is responsible for exploring the hyper parameter space while dynamically identifying the most suitable network architecture for the data being processed.

The experiments conducted allowed us to compare this approach with classical such as Grid Search, Random Search, highlighting a significant improvement in terms of accuracy, execution time, and robustness of the generated models.

Keywords : Metaheuristics, Genetic Algorithm, Deep Learning, MLP, DNN, CNN, LSTM, Grid Search, Random Search, Bayesian optimization.

ملخص

يندرج هذا البحث في إطار تحسين أداء نماذج التعلم الآلي، بالاعتماد على استخدام الخوارزميات الجينية (GA). الهدف الرئيسي هو اقتراح مقاربة تحسين مشتركة لكل من الضبط التلقائي للمعاملات الفائقة (hyperparamètres) واختيار النموذج الأمثل تلقائياً بين أربعة نماذج من التعلم العميق (DL)، وهي: CNN، DNN، MLP، وLSTM.

يعتمد مسار التحسين المقترح على نمذجتين:

- نمذجة أحادية الهدف (تستهدف تحسين معدل الاسترجاع - Recall)
- نمذجة متعددة الأهداف (تركز على درجة F1-score - F1)،

وذلك لتوجيه البحث نحو نماذج ذات قدرة عالية على التعميم.

تُكلّف الخوارزمية الجينية باستكشاف فضاء المعاملات الفائقة، مع تحديد الهيكل الأمثل للشبكة العصبية المتوافق مع طبيعة البيانات المعالجة.

وقد سمحت التجارب المنجزة بمقارنة هذه المقاربة مع طرق التحسين التقليدية مثل:

- البحث الشبكي (Grid Search)
- البحث العشوائي (Random Search)
- التحسين البايزي (Bayesian Optimization)

وأظهرت النتائج تحسناً ملحوظاً من حيث الدقة، سرعة التنفيذ، ومتانة النماذج الناتجة.

الكلمات المفتاحية: الميتاهيروسطيك، الخوارزميات الجينية، التعلم العميق، CNN، LSTM، DNN، MLP، البحث الشبكي، البحث العشوائي، التحسين البايزي.

TABLE DES MATIÈRES

INTRODUCTION GÉNÉRALE 15
Chapitre I Étude bibliographique 18
1.1 Introduction 18
1.2 Machine Learning 18
1.3 Sélection des Modèles 23
1.4 Optimisation des hyper paramètres 25
1.5 Modèles DL 30
1.6 Heuristiques et méta heuristiques 37
1.7 Conclusion 40
Chapitre II Contribution 41
2.1 Introduction 41
2.2 Présentation des architectures de DL utilisées 42
2.3 Conclusion 50
Chapitre III Implémentation ET Résultats 51
3.1 Introduction 51
3.2 Outils d'implémentation 51
3.3 Environnement Expérimental 55
3.4 CONCLUSION : 61
CONCLUSION GÉNÉRALE et Perspectives 62

LISTE DES FIGURES

Figure 1 : Types de machine learning [2]. 19
Figure 2 : Processus de classification supervisée. 21
Figure 3 : Réglage des hyper paramètres [10] 26
Figure 4 : Les approches de réglages des hyper paramètres 27
Figure 5 : Architectures des réseaux de neurones du Deep Learning [14] 30
Figure 6 : Architecture de MLP pour classification binaire 32
Figure 7 : Architecture de DNN pour classification binaire 33
Figure 8 :Architecture de CNN pour classification binaire 34
Figure 9 :Architecture de LSTM pour classification binaire 35
Figure 10 : Fonctionnement d'un Algorithme Génétique (GA) simple [37] 40
Figure 12 : Architecture générale des modèles DL 42
Figure 13 : Architecture de modèle MLP 44
Figure 14 : Architecture de modèle DNN 45
Figure 15 : Architecture de modèle LSTM 46
Figure 16 : Architecture de CNN 47
Figure 21 : Environnement logiciel 53

LISTE DES TABLEAUX

Table 1 : Comparaison entre les types d'apprentissage 21
Table 2 : Comparaison des méthodes HPO 30
Table 4 : Comparaison entre les modèles (ML) CNN, DNN, MLP, LSTM 36
Table 6 : Comparaison descriptive des architectures neuronales CNN, LSTM, DNN et MLP 49
Table 11 ::Description des Outils d 'implémentation 55
Table 12 :Evironnement matériel et logiciel 56
Table 13 :Structure du jeu de données 58

ACRONYMES

ANN: Artificial Neural Network

Auto ML: Automated Machine Learning

CNN: Convolutional Neural Network

DL: Deep Learning

DNN: Deep Neural Network

FN: False Negative

FP: False Positive

GA: Genetic Algorithm

GS: Grid Search

HPO: Hyper parameter Optimization

IA: Intelligence Artificielle

LSTM: Long Short-Term Memory

ML: Machine Learning

MLP: Multi-Layer Perceptron

PSO : Particle Swarm Optimization

ReLU: Rectified Linear Unit

RNN: Recurrent Neural Network

ROC: Receiver Operating Characteristic

RS: Random Search

SA: Simulated Annealing

TN: True Negative

TP: True Positive

INTRODUCTION GÉNÉRALE

L'essor spectaculaire de l'intelligence artificielle (Intelligence Artificiel, IA), et plus particulièrement de l'apprentissage automatique (Machine Learning, ML), a profondément transformé de nombreux secteurs tels que la santé, la finance, le cyber sécurité, les transports et l'industrie. Ces avancées s'appuient sur des modèles capables d'extraire automatiquement des informations pertinentes à partir de vastes ensembles de données, ouvrant la voie à des systèmes plus intelligents et autonomes. Cependant, les performances de ces modèles dépendent fortement d'un processus d'optimisation rigoureux, qui englobe plusieurs étapes clés : la sélection des caractéristiques, le réglage des hyper paramètres, l'ajustement des poids internes, ainsi que la sélection finale du modèle le plus performant. Parmi ces étapes, l'optimisation des hyper paramètres et la sélection automatique du meilleur modèle représentent des enjeux majeurs, notamment dans le cadre de l'apprentissage profond (deep learning).

D'une part, le réglage des hyper paramètres joue un rôle central dans la stabilité, la précision, la capacité de généralisation et la rapidité d'apprentissage des modèles. Cette tâche devient particulièrement complexe dans le cas des architectures profondes telles que le MLP (Multi-Layer Perceptron), le DNN (Deep Neural Network), le CNN (Convolutional Neural Network) ou le LSTM (Long Short-Term Memory), qui comportent un grand nombre de paramètres sensibles. Les approches classiques comme : Grid Search, Random Search et l'optimisation Bayésienne, sont souvent utilisées pour cette fin. Néanmoins, ces méthodes sont limitées par leur inefficacité dans des espaces de recherche vastes et non convexes, et leur incapacité à tirer parti des évaluations précédentes pour guider intelligemment la recherche [1].

D’autre part, le choix du meilleur modèle parmi plusieurs candidats entraînés est souvent réalisé manuellement ou via des comparaisons empiriques, ce qui augmente le risque de sous-optimisation et ralentit le processus de développement. Or, automatiser cette phase peut considérablement améliorer la qualité et la rapidité du cycle d’apprentissage.

Dans ce contexte, les algorithmes génétiques (AG) s’imposent comme une approche prometteuse. Inspirés du processus de sélection naturelle, ils permettent d’explorer efficacement des espaces de solutions complexes, d’éviter les minimas locaux et d’atteindre des configurations quasi-optimales, même en présence de non-linéarités fortes et de haute dimensionnalité. Appliqués à l’apprentissage profond, les AGs offrent la possibilité non seulement de régler automatiquement les hyper paramètres, mais également de sélectionner le modèle le plus performant, rendant le processus global plus intelligent, adaptatif et moins coûteux.

À partir de ces constats, la problématique de ce travail peut être formulée comme suit : Comment l’Algorithme Génétique (AG) peut-il contribuer à la fois à l’optimisation des hyper paramètres et à la sélection automatique du modèle de deep learning le plus performant (parmi MLP, DNN, CNN, LSTM), en surpassant les méthodes classiques telles que Grid Search, Random Search et Optimisation Bayésienne en termes de précision, de rapidité, de convergence et de coût computationnel ?

Ce projet vise à répondre à cette problématique en poursuivant les objectifs suivants :

- Étudier le rôle global des méta-heuristiques, en particulier des algorithmes génétiques, dans l’optimisation des modèles de deep learning ;
- Mettre en œuvre une approche combinée permettant d’optimiser les hyper paramètres et de sélectionner automatiquement le modèle le plus adapté à une tâche donnée ;
- Comparer les performances de cette approche avec celles obtenues par les méthodes classiques d’optimisation :(Grid Search, Random Search et Optimisation Bayésienne) ;

- Évaluer l’efficacité de l’AG sur différentes architectures de réseaux profonds, en particulier les MLP, DNN, CNN et LSTM, en tenant compte de critères tels que la précision, le temps de convergence et la charge computationnelle.

Pour atteindre les objectifs fixés, ce mémoire est structuré en trois chapitres principaux :

- Le premier chapitre présente une étude bibliographique portant sur l’apprentissage automatique, les modèles de deep learning, ainsi que les problématiques liées à l’optimisation des hyper paramètres et à la sélection de modèles. Il s'intéresse également aux approches méta-heuristiques, notamment l’algorithme génétique (AG).
- Le deuxième chapitre est consacré à la modélisation des problèmes d’optimisation des hyper paramètres (HPO) et de sélection de modèles d’apprentissage profond, en proposant une solution basée sur l’algorithme génétique.
- Le troisième et dernier chapitre** est dédié à l’implémentation des algorithmes étudiés, ainsi qu’aux tests expérimentaux réalisés pour évaluer leur performance.

Ce travail se conclut par une synthèse des résultats obtenus, suivie d’une ouverture sur les perspectives futures.

Chapitre I

Étude bibliographique

1.1 Introduction

Ce chapitre présente les fondements théoriques sur lesquels repose notre travail. Il commence par un aperçu global de la machine learning, en abordant ses principes généraux ainsi que les principales architectures de réseaux de neurones telles que les MLP, DNN, CNN et LSTM, largement utilisées dans les systèmes d'apprentissage automatique modernes. Nous traitons ensuite de l'optimisation des hyper paramètres, une étape clé dans le processus d'entraînement des modèles, en passant en revue les méthodes existantes et leur efficacité comparative. La sélection des modèles constitue également une partie importante de cette étude, car elle permet d'identifier les configurations les plus performantes selon des critères précis. Enfin, le chapitre s'intéresse au méta heuristique, des approches d'optimisation inspirées de processus naturels, qui seront exploitées dans ce projet pour améliorer la performance globale du système. Ce cadre théorique offre ainsi une base solide pour la suite du mémoire, notamment pour le développement de notre approche.

1.2 Machine Learning

Le Machine Learning (ML), ou apprentissage automatique, est un domaine clé de l'intelligence artificielle qui repose sur le développement d'algorithmes et de modèles statistiques permettant aux systèmes informatiques d'apprendre automatiquement à partir des données, sans être explicitement programmés. En identifiant des motifs récurrents et en tirant des inférences, ces systèmes deviennent capables de résoudre des problèmes complexes, d'optimiser des processus, et de prendre des décisions intelligentes. Leur performance s'améliore au fil du temps à

mesure qu'ils sont exposés à de nouvelles données, ce qui en fait un outil particulièrement puissant pour des tâches telles que la sélection de solutions optimales, la prédiction de comportements, ou encore la reconnaissance de schémas dans des environnements incertains. [1]

1.2.2 Types de machine learning

L'apprentissage automatique peut être catégorisé en plusieurs types selon la nature des données disponibles et les objectifs visés. Les trois formes principales sont celles de la (figure 1) : l'apprentissage supervisé, l'apprentissage non supervisé et l'apprentissage par renforcement.

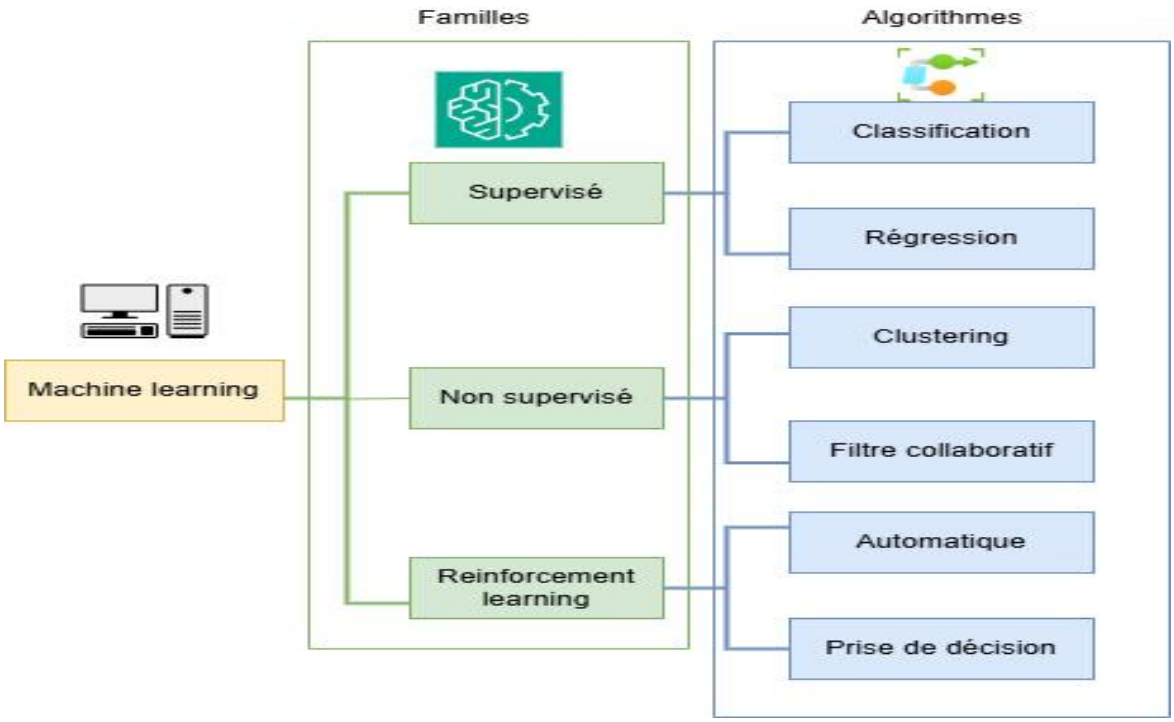

Figure 1 **:** Types de machine learning [2].

Dans l'apprentissage supervisé, l'algorithme apprend à partir d'un ensemble de données étiquetées, c'est-à-dire que chaque exemple de données est associé à une réponse ou à une sortie correcte. L'objectif est alors de construire un modèle capable de généraliser cette connaissance pour prédire la sortie attendue de nouvelles données jamais vues. Ce type d'apprentissage est largement utilisé dans les tâches de classification (comme la détection de courriels indésirables ou la reconnaissance d'images) ou de régression (comme la prévision du prix d'un logement ou la

température). L'efficacité des algorithmes supervisés dépend fortement de la qualité et de la quantité des données d'entraînement disponibles, car des données mal étiquetées ou insuffisantes peuvent considérablement nuire à la performance du modèle final. [3]

L'apprentissage non supervisé, quant à lui, se caractérise par l'absence d'étiquettes dans les données. L'algorithme doit alors explorer la structure intrinsèque des données pour y découvrir des regroupements, des schémas ou des représentations significatives. Il est utilisé principalement pour des tâches telles que le clustering, où l'on cherche à regrouper les données selon des similarités naturelles, ou la réduction de dimensionnalité, qui vise à simplifier les données tout en conservant l'essentiel de leur information. Ce type d'apprentissage est particulièrement utile pour l'analyse exploratoire, la détection d'anomalies ou encore la préparation des données en amont d'un apprentissage supervisé. Cependant, l'évaluation des performances reste délicate, car l'absence de vérité terrain rend difficile la validation des résultats. [4]

Enfin**,** l'apprentissage par renforcement repose sur un principe totalement différent. Il s'agit d'un paradigme où un agent autonome apprend à prendre des décisions en interagissant avec un environnement. À chaque action qu'il effectue, l'environnement lui fournit une rétroaction sous forme de récompense (positive ou négative). L'objectif de l'agent est alors d'apprendre une stratégie (ou politique) optimale qui maximise la somme des récompenses reçues sur le long terme. Ce type d'apprentissage est particulièrement adapté aux situations séquentielles, comme les jeux de stratégie, la robotique mobile ou encore certaines applications financières. Toutefois, cette approche nécessite généralement un très grand nombre d'interactions, ce qui peut engendrer un coût computationnel élevé et une complexité de mise en œuvre non négligeable. [5]

Ainsi, ces trois formes d'apprentissage répondent à des problématiques différentes mais complémentaires dans le champ du Machine Learning (tableau 1). Le choix du type d'apprentissage dépend de la nature des données disponibles, des ressources, ainsi que des objectifs de l'application ciblent. Dans le cadre de ce mémoire, notre attention se portera principalement sur la classification supervisée, dont le processus sera détaillé dans la section suivante.

Type d'apprentissage	Objectif	Applications typiques	Avantages	Inconvénients
Supervisé	Prédire des sorties connues	Classification, régression, détection de spam	Haute précision si beaucoup de données	Besoin d'un grand volume de données étiquetées
Non supervisé	Découvrir des structures ou regroupements	Clustering, réduction de dimension, détection d'anomalies	Pas besoin d'étiquetage, utile pour explorer	Résultats difficiles à évaluer ou interpréter
Renforcement	Maximiser une récompense cumulée	Jeux, robotique, trading, systèmes autonomes	Adaptation dynamique à l'environnement	Entraînement long, environnement complexe à modéliser

Table 1 : Comparaison entre les types d'apprentissage

1.2.3 Processus de classification supervisée

Le développement d'un modèle performant en apprentissage supervisée repose sur un enchaînement rigoureux d'étapes successives, allant de la collecte des données à la sélection finale du meilleur modèle .Chaque phase joue un rôle fondamental dans la qualité des prédictions produites. Le schéma présenté illustre ce processus global.
[6]

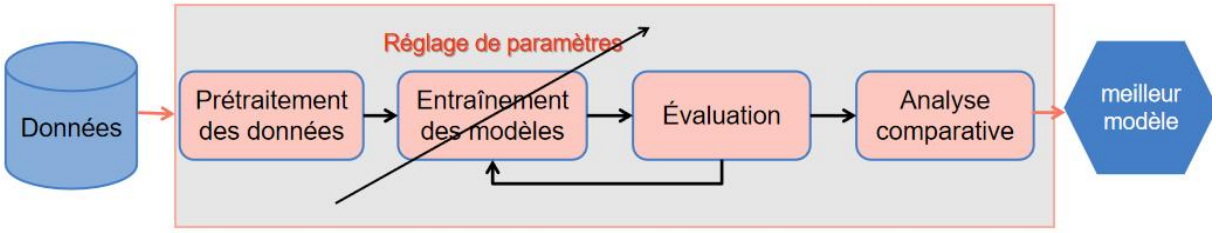

Figure 2**:** Processus de classification supervisée.

a. Données

La première étape consiste à collecter des données pertinentes, qui serviront de base à l'apprentissage du modèle. Ces données doivent être représentatives du phénomène étudié, suffisamment volumineuses, diversifiées et étiquetées avec soin. En effet, la qualité des données conditionne directement la capacité du modèle à généraliser et à produire des résultats fiables sur de nouvelles instances.

b. Prétraitement des données

Une fois les données disponibles, un travail de prétraitement est indispensable pour les nettoyer, les transformer et les préparer à l'entraînement. Cette phase inclut généralement la gestion des valeurs manquantes, la détection et le traitement des valeurs aberrantes, la normalisation ou standardisation des variables, le codage des attributs catégoriels, ainsi que la division du jeu de données en sous-ensembles d'apprentissage, de validation et de test. Ce travail assure que les données sont dans un format exploitable et cohérent, favorisant ainsi la stabilité et la performance du modèle.

c. Entraînement des modèles

L'entraînement des modèles consiste à ajuster automatiquement les paramètres internes du modèle (poids, biais, etc.) à partir des données d'apprentissage. Le but est de minimiser une fonction de perte qui mesure l'écart entre les prédictions du modèle et les valeurs réelles. Ce processus permet au modèle d'apprendre les relations sous-jacentes aux données, de façon à pouvoir ensuite prédire correctement de nouvelles observations.

d. Réglage des hyper paramètres

Parallèlement à l'entraînement, le réglage des hyper paramètres est une étape cruciale. Contrairement aux paramètres appris automatiquement, les hyper paramètres sont fixés manuellement ou via des techniques d'optimisation. Leur ajustement permet d'améliorer significativement la performance du modèle. Des méthodes telles que la recherche par grille (Grid Search), la recherche aléatoire (Random Search), Optimisation Bayésienne ou le méta heuristique sont couramment utilisées pour identifier la meilleure combinaison d'hyper paramètres.

e. Évaluation des performances

Une fois les modèles entraînés, leur performance est évaluée sur un jeu de données de test indépendant. Cette étape permet de mesurer la capacité de généralisation du modèle à travers des métriques comme la précision, le rappel, la F-mesure ou l'aire sous la courbe ROC. Une évaluation rigoureuse est essentielle pour s'assurer que le modèle ne se contente pas de mémoriser les données d'apprentissage, mais qu'il est apte à fonctionner correctement sur des données nouvelles.

f. Analyses comparative

À cette étape, les performances de plusieurs modèles sont comparées en fonction des résultats obtenus à l'évaluation. Le modèle présentant les meilleurs scores sur les indicateurs choisis est alors sélectionné comme le plus performant. Ce processus implique souvent l'analyse croisée des résultats sur l'ensemble de validation et l'ensemble de test, afin de garantir un choix éclairé et éviter tout risque de sur apprentissage.

g. Meilleur modèle retenu

Le modèle final retenu est celui qui démontre la meilleure capacité à généraliser et à prédire avec précision sur des données inconnues. Il peut être ré-entraîné sur l'ensemble complet des données (entraînement + validation) pour bénéficier d'une quantité maximale d'information avant déploiement. Ce modèle est ensuite utilisé en production pour effectuer des prédictions en situation réelle. [7]

1.3 Sélection des Modèles

La sélection du modèle constitue une étape déterminante avant toute tentative d'optimisation. Le modèle ML choisi doit être en adéquation avec la nature des données, la complexité du problème et les ressources disponibles. Un modèle mal adapté, qu'il soit trop simple ou excessivement complexe, risque soit de sous-apprendre (manque de capacité), soit de sur apprendre (perte de généralisation), impactant négativement la qualité des prédictions [8] [9]

En se focalisant dans notre étude sur la sélection des modèles d'apprentissage profond, la sélection d'un modèle DL vise à :

- Identifier une architecture offrant un bon compromis entre expressivité et capacité de généralisation ;
- Assurer une adéquation entre la structure du réseau et les caractéristiques des données (type, volume, bruit) ;
- Réduire les coûts en temps de calcul et en ressources matérielles, notamment lors de l'entraînement ;
- Faciliter, lorsque cela est nécessaire, l'interprétable et la robustesse du modèle.

1.3.1 Méthodes de sélection

En général, la sélection d'un modèle ML peut s'effectuer selon deux approches complémentaires :

- Approche empirique manuelle : fondée sur l'expertise, l'expérience et l'ajustement progressif via essais et erreurs. Cette méthode reste courante mais peut être coûteuse en temps et peu exhaustive.
- Approche systématique et automatique : faisant recours à des méthodes d'exploration et d'optimisation de l'espace de recherche et hyper paramètres, telles que la recherche par grille, la recherche aléatoire, les optimisations bayésiennes, ou des algorithmes méta heuristiques. Ces techniques permettent d'automatiser la découverte de modèles adaptés tout en limitant les biais humains.

1.3.2 Critères de sélection

La sélection doit tenir compte d'un ou de plusieurs de ces critères [8] [9]**:**

a. Performance prédictive

La qualité d'un modèle est avant tout mesurée par ses performances sur des données de validation ou de test, au moyen de métriques pertinentes (accuracy, F1-score,

erreur quadratique moyenne, etc.), adaptées à la nature de la tâche (classification, régression, etc.).

b. Complexité et capacité du modèle

Le modèle doit posséder une capacité suffisante pour apprendre les représentations complexes inhérentes aux données, tout en évitant la sur complexité qui mène au sur apprentissage. La profondeur, le nombre de neurones et la connectivité sont autant d'éléments à calibrer selon la difficulté du problème.

c. Nature des données

La sélection doit tenir compte du format et des spécificités des données en entrée. Par exemple, un modèle entièrement connecté peut convenir à des données tabulaires, tandis que certaines structures nécessitent des adaptations pour gérer des données séquentielles ou spatiales, même si ici l'attention reste sur les ANN en général.

d. Généralisation

La robustesse d'un modèle est évaluée par sa capacité à maintenir des performances satisfaisantes sur des données non vues. Cela implique l'usage de stratégies de régularisation (dropout, pénalités L2, early stopping) et des techniques d'évaluation rigoureuses comme la validation croisée.

e. Contraintes opérationnelles

Les ressources matérielles (mémoire, temps d'entraînement), la vitesse d'inférence et la complexité d'implémentation sont des facteurs essentiels, surtout dans des contextes contraints (temps réel, embarqué, production).

1.4 Optimisation des hyper paramètres

L'optimisation des hyperparamètres (HPO) désigne le processus systématique de recherche des meilleures valeurs possibles pour les hyperparamètres d'un modèle d'apprentissage automatique, dans le but de maximiser ses performances sur des données de validation. Contrairement aux paramètres internes du modèle (tels que les poids et biais), qui sont ajustés automatiquement par rétropropagation pendant l'entraînement, les hyperparamètres sont fixés avant l'apprentissage et influencent

directement le comportement, la structure et la performance du modèle.La figure 6 présente la fonction d'optimisation des hyper paramètres qui prend en entrée ces paramètres de configuration externes et applique diverses stratégies pour identifier leur combinaison optimale.

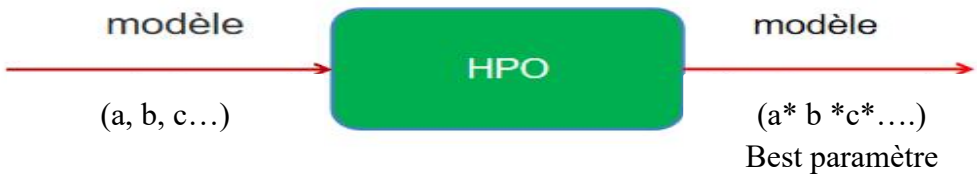

Figure 3 : Réglage des hyper paramètres [10]

Dans le contexte des réseaux de neurones artificiels (ANN), le choix des hyperparamètres — tels que le taux d'apprentissage, la taille du batch, le nombre de couches ou de neurones — joue un rôle déterminant dans :

- la capacité d'apprentissage du modèle,
- la stabilité de la convergence durant l'entraînement,
- et la généralisation sur des données inconnues.

Cependant, le problème de HPO est hautement non trivial en raison de plusieurs facteurs :

- la non-linéarité de la fonction objectif (souvent l'erreur sur un ensemble de validation),
- la dimensionnalité élevée de l'espace de recherche,
- et les interdépendances complexes entre les différents hyperparamètres.

Face à cette complexité, une configuration inadéquate peut compromettre l'efficacité du modèle, d'où l'importance cruciale de bien ajuster ces paramètres. L'objectif central du HPO est donc de trouver la configuration optimale permettant au modèle de produire des prédictions fiables, tout en évitant le sous-apprentissage ou le sur-apprentissage

1.4.2 Méthodes d'optimisation des hyper paramètres

L'optimisation des hyperparamètres peut être abordée selon différentes stratégies, allant des méthodes empiriques simples aux techniques automatisées sophistiquées. Ces approches diffèrent par leur efficacité, leur complexité et leur capacité à explorer de grands espaces de recherche. On distingue généralement deux grandes familles : les méthodes manuelles et les approches automatiques [11][34][35].

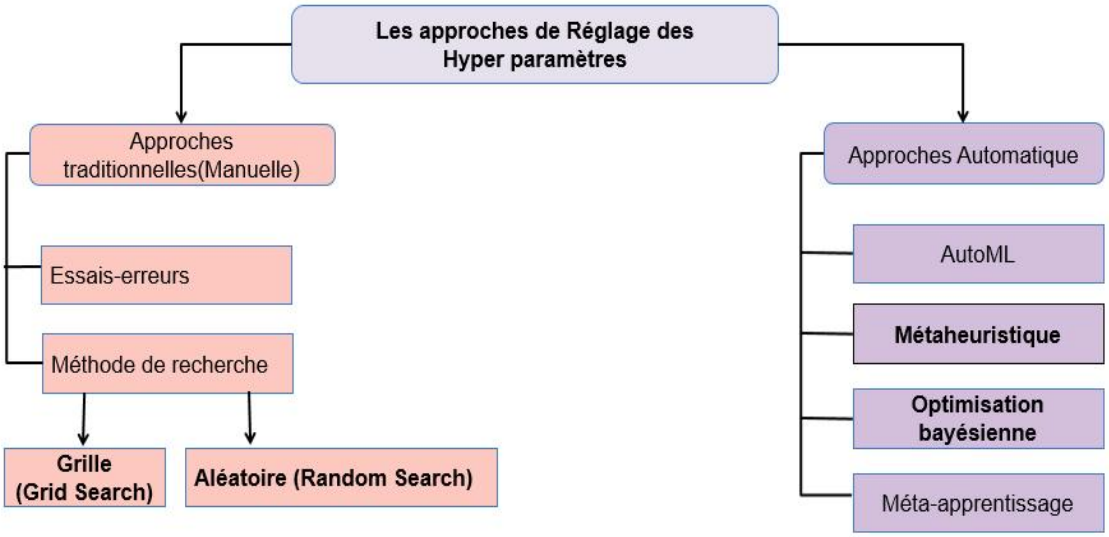

Figure 4 : Les approches de réglages des hyper paramètres

a. Approches manuelles : Ces techniques sont basées sur l'intervention humaine et ne reposent pas sur une stratégie d'optimisation formelle.

- Essais-erreurs (Trial-and-Error) **:** Il s'agit d'une méthode empirique qui consiste à tester manuellement diverses configurations de manière ad hoc. Bien qu'intuitive, cette approche est souvent inefficace pour les modèles complexes, car elle repose sur des choix arbitraires, sans exploration systématique ni garantie de performance.
- Recherche en grille (Grid Search) : Cette méthode évalue toutes les combinaisons possibles d'un ensemble de valeurs prédéfinies. Elle est facile à implémenter mais devient rapidement coûteuse en temps de calcul dès que le nombre de dimensions augmente.
- Recherche aléatoire (Random Search) : Elle consiste à tirer au hasard des configurations dans l'espace des hyperparamètres. Moins

exhaustive que Grid Search, elle s'avère pourtant souvent plus efficace, en particulier lorsqu'un petit nombre de paramètres influence fortement la performance.

b. <u>Approches automatiques :</u> Ces approches reposent sur des algorithmes capables d'explorer l'espace de recherche avec peu ou pas d'intervention humaine.

- <u>AutoML (Automated Machine Learning)</u> : Ces systèmes automatisent l'ensemble du processus de modélisation : sélection des modèles, optimisation des hyperparamètres, validation croisée, etc. Ils permettent un déploiement rapide mais requièrent des ressources computationnelles importantes.

- <u>Optimisation bayésienne</u> : Cette technique probabiliste modélise la fonction objectif à optimiser, généralement via des processus gaussiens. Une fonction d'acquisition guide la recherche en équilibrant exploration (recherche de nouvelles régions) et exploitation (raffinement autour de bonnes solutions). Elle est efficace avec peu d'échantillons, mais sensible à la complexité du modèle et au bruit.
- <u>Méta-apprentissage :</u> Le méta-apprentissage (ou *learning to learn*) vise à apprendre à partir de plusieurs expériences antérieures pour adapter dynamiquement la stratégie d'optimisation sur de nouvelles tâches. Il repose souvent sur des réseaux récurrents ou à mémoire externe capables d'encoder des séquences d'expériences passées.
- <u>Méta-heuristiques</u> : Il s'agit d'algorithmes d'optimisation stochastiques inspirés de processus naturels (évolution, physique, comportement animal, etc.). Leur capacité à gérer des espaces de recherche non convexes, discrets ou hybrides en fait un choix adapté aux problèmes complexes de HPO, en particulier pour:
  - explorer efficacement un espace de solutions vaste et irrégulier ;
  - combiner la recherche de modèles et d'hyperparamètres (optimisation conjointe) ;
  - traiter des objectifs multiples (précision, temps, robustesse...).

1.4.3 Comparaison des méthodes d'optimisation des hyper paramètres

Le tableau suivant propose une synthèse comparative des principales approches décrites précédemment pour résoudre ce problème, en mettant en évidence leurs principes, avantages et limites.

Méthode HPO	Principe	Avantages	Limites
Essais-Erreurs (Manuelle)	Réglage empirique basé sur l'intuition ou l'expérience de l'utilisateur	Simple à mettre en œuvre ; adaptée aux petits modèles ou jeux de données	Non systématique, inefficace sur espaces complexes ou dimensionnels
Grid Search	Recherche exhaustive sur une grille discrète de combinaisons	Facile à implémenter ; reproductible	Très coûteuse en temps de calcul (explosion combinatoire)
Random Search	Échantillonnage aléatoire de l'espace des hyperparamètres	Plus efficace que Grid Search sur espaces vastes ; flexible	Non optimal ; ne garantit pas une couverture suffisante
Optimisation Bayésienne	Modélisation probabiliste de la fonction objectif ; guide l'exploration	Échantillonnage intelligent ; efficace avec peu d'itérations	Implémentation plus complexe ; sensible à la dimensionnalité
Métaheuristiques	Algorithmes bio-inspirés (GA, PSO, SA) explorant de grands espaces non linéaires	Robuste ; adapté aux fonctions non convexes et aux modèles profonds	Coût computationnel élevé ; convergence parfois lente
AutoML	Pipeline entièrement automatisé de sélection de modèles et HPO	Très utile pour les non-experts ; pipeline complet automatisé	Moins de contrôle sur les choix algorithmiques ; nécessite ressources

			importantes
Méta-apprentissage	Le système apprend à optimiser les HPO à partir d'expériences antérieures	Adaptatif, transfert entre tâches ; efficace pour problèmes récurrents	Complexité algorithmique élevée ; dépend des méta-données disponibles

Table 2: Comparaison des méthodes HPO

1.5 Modèles DL

Les modèles de Deep Learning sont basés sur des réseaux de neurones profonds, c'est-à-dire constitués de plusieurs couches cachées, leur permettant de capturer des relations complexes et non linéaires entre les données d'entrée et de sortie. Grâce à leur capacité de généralisation, ces modèles se révèlent particulièrement performants aussi bien pour les tâches d'apprentissage supervisé que non supervisé, en particulier lorsqu'ils sont entraînés sur de grands volumes de données [13].

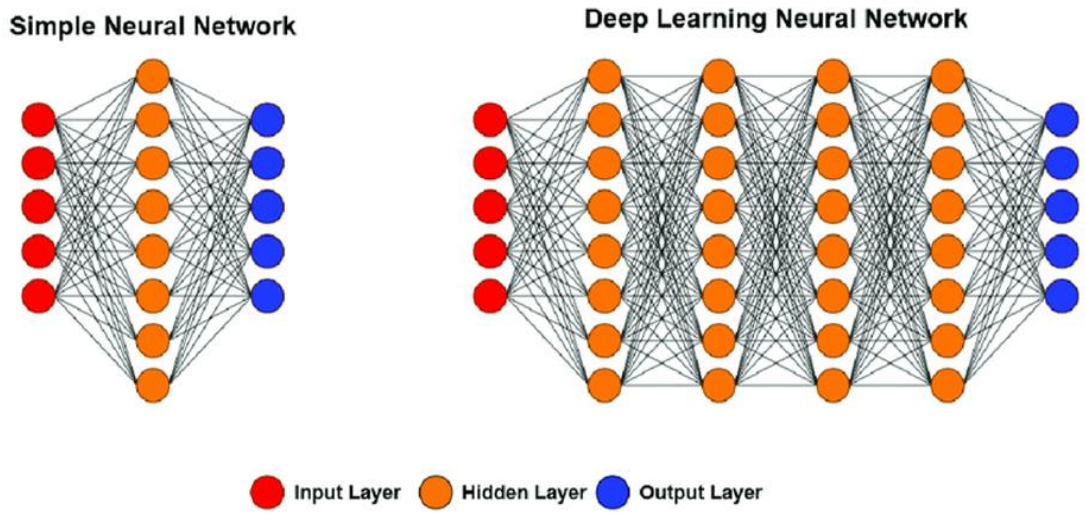

Figure 5 : Architectures des réseaux de neurones du Deep Learning [14]

On distingue plusieurs architectures de réseaux de neurones, généralement classées selon leur structure et le type de données qu'elles traitent .Parmi les plus utilisées, on retrouve :

- Le MLP (Multi-Layer Perceptron), adapté aux données tabulaires ;
- Le DNN (Deep Neural Network), une version plus profonde des MLP ;
- Le CNN (Convolutional Neural Network), performant pour les données structurées spatialement comme les images ;
- Le LSTM (Long Short-Term Memory), conçu pour le traitement des données séquentielles et temporelles.

Ces architectures seront détaillées dans la section suivante :

1.5.1 MLP (Multi-Layer Perceptron)

Le MLP (Multilayer Perceptron) est l'architecture la plus simple des réseaux de neurones. Il est composé :

- D'une couche d'entrée qui reçoit les données d'entrée ;
- D'une ou plusieurs couches cachées ;
- D'une couche de sortie.

Chaque neurone réalise deux opérations principales : une combinaison linéaire suivie d'une fonction d'activation non linéaire :

$$z^{l} = w^{l} * a^{l-1} + b^{l}$$

$$a^{l} = f(z^{l})$$

Où :

- W est la matrice des poids de la couche l,
- b est le vecteur des biais,
- a est la sortie de la couche précédente,
- f est une fonction d'activation (par exemple : ReLU, Sigmoïde, Tanh). [15]

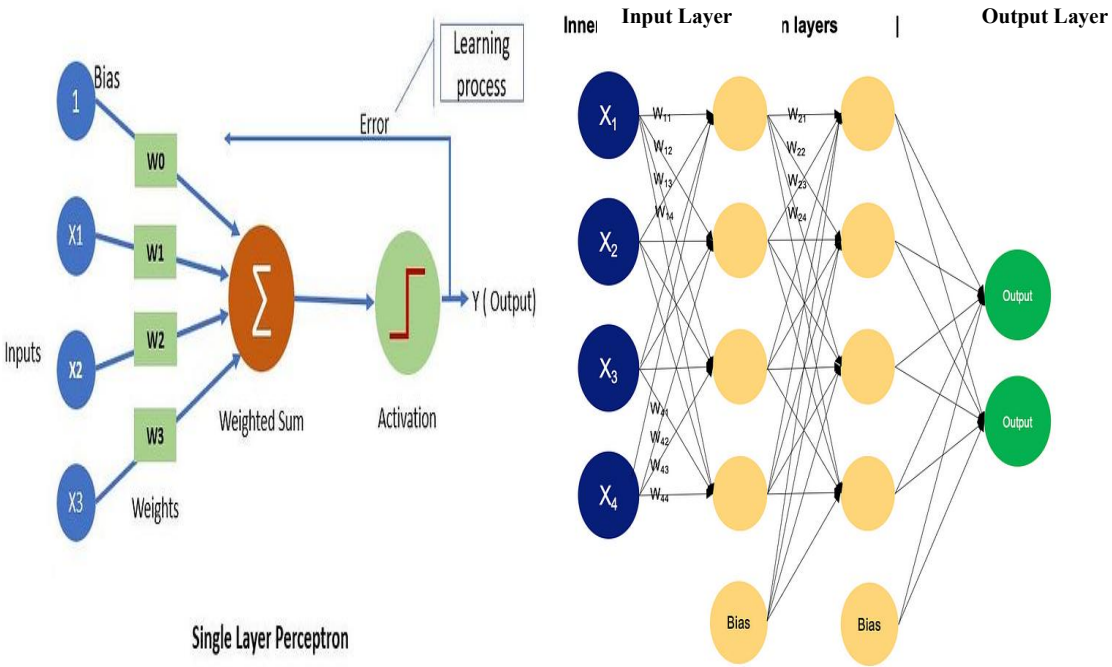

Figure 6 : Architecture de MLP pour classification binaire

1.5.2 DNN (Deep Neural Network)

Le DNN (Deep Neural Network) est une extension du MLP avec un plus grand nombre de couches cachées. Il permet de capturer des relations non linéaires plus profondes et d'extraire des représentations de plus haut niveau.

Le fonctionnement mathématique reste similaire à celui du MLP :

$$a^0 = x \text{ (entrée)}$$

$$z^l = w^l * a^{l-1} + b^l \quad \text{(pour chaque couche l).}$$

$$a^l = f(z^l)$$

Les DNN nécessitent un réglage précis des hyper paramètres (profondeur, taille des couches, taux d'apprentissage, etc.) et sont sensibles au sur apprentissage.

MLP **DNN**

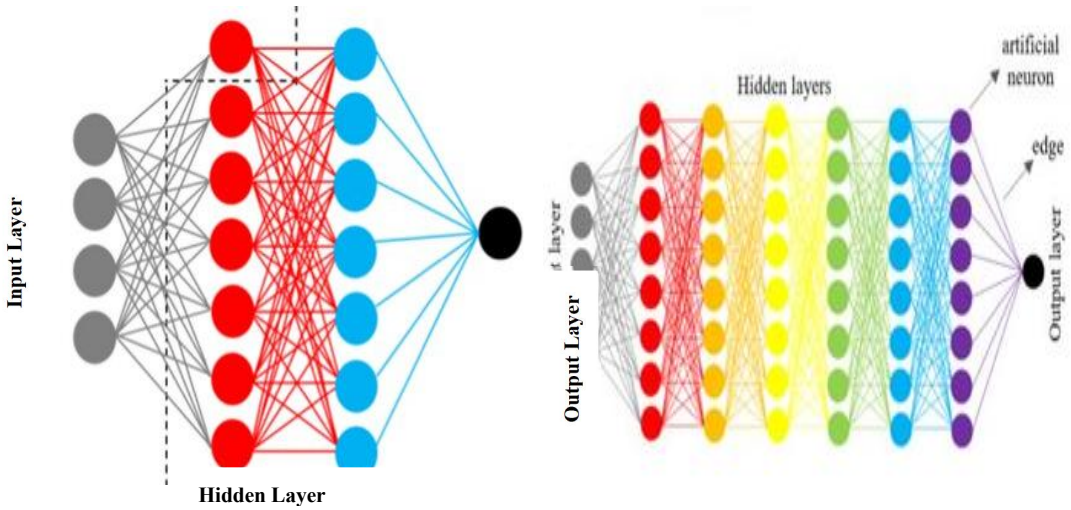

Figure 7 : Architecture de DNN pour classification binaire

1.5.3 CNN (Convolutional Neural Network)

Les CNN (Convolutional Neural Network) sont conçus pour les données structurées spatialement, comme les images. L'architecture se compose de :

- Couches de convolution, qui extraient des caractéristiques locales.
- Couches de pooling, qui réduisent la dimension spatiale.
- Couches entièrement connectées, en fin de réseau pour la décision.

❖ Formule de convolution 2D :

$$S(i, j) = (X * K)(i, j) = \sum_m \sum_n X(i + m, j + n) \cdot K(m, n)$$

Où

X est l'entrée (ex. : une image),

K est le filtre (ou noyau de convolution),

S (i, j) est la valeur en position (i, j) de la sortie.

Pooling max (exemple 2×2):

P (i, j) = max{S (2i, 2j), S (2i + 1, 2j), S (2i, 2j + 1), S (2i + 1, 2j + 1)}

Les CNN sont efficaces pour la reconnaissance d'images, mais également pour certaines tâches en cyber sécurité (ex. : analyse de séquences binaires, détection d'anomalies dans des matrices de trafic réseau).

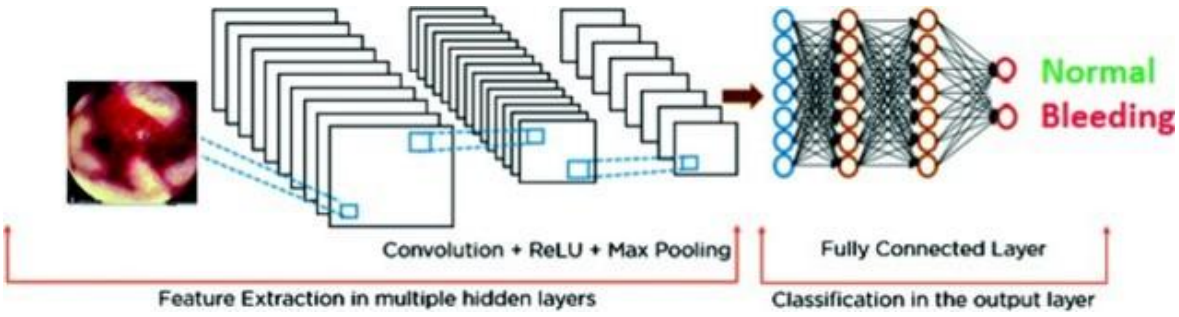

Figure 8 :Architecture de CNN pour classification binaire

1.5.4 LSTM (Long Short-Term Memory)

Les LSTM sont une extension des réseaux de neurones RNN conçue pour le traitement de données séquentielles (texte, séries temporelles, logs, etc.). Grâce à des portes de régulation, ils contrôlent le flux d'informations et évitent le problème de disparition du gradient rencontré dans les RNN classiques.

Voici les équations de base d'une cellule LSTM à l'instant t

$$f_t = \sigma\,(W_f \cdot [h_{\{t-1\}}, x_t] + b_f) \quad \text{(porte d'oubli)}$$

$$i_t = \sigma\,(W_i \cdot [h_{\{t-1\}}, x_t] + b_i) \quad \text{(porte d'entrée)}$$

$$\{\widetilde{C}\}_t == tanh\big(W_C \cdot \big[h_{\{t-1\}}, x_t\big] + b_C\big) \text{ (état candidat)}$$

$$C_{\{t\}} = f_t . C_{\{t-1\}} + i_t . \widetilde{\{C\}_t} \quad \text{(Mise à jour de l'état de la cellule)}$$

$$\sigma = \big(W_\sigma \cdot \big[h_{\{t-1\}}, x_t\big] + b_\sigma\big) \text{ (porte de sortie)}$$

$$h_{\{t\}} = \sigma_t . \tanh\big(C_{\{t\}}\big) \text{ (Sortie de la cellule)}$$

Où :

- xt : vecteur d'entrée à l'instant t,
- ht : sortie de la cellule,
- Ct : état de la mémoire,
- σ : fonction sigmoïde,
- . : produit élément par élément.

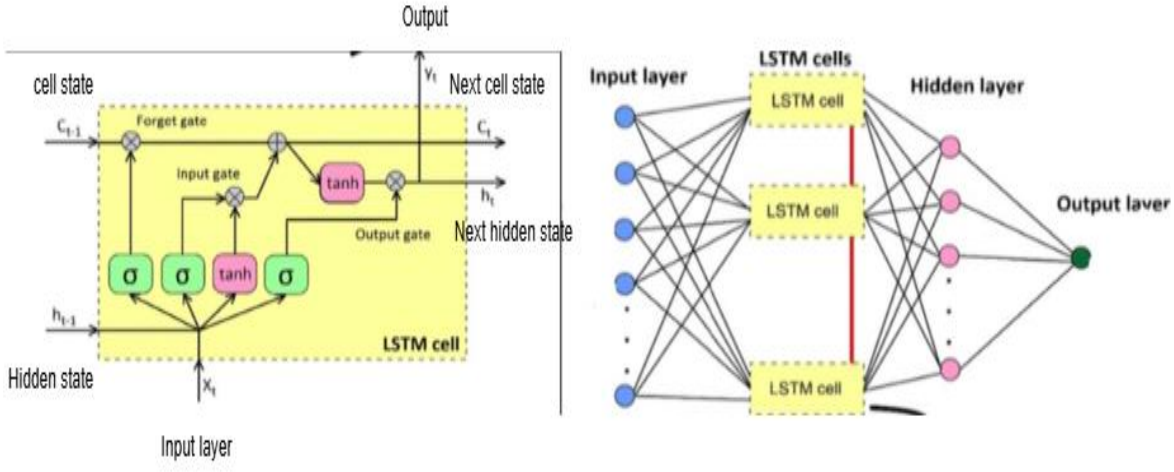

Figure 9 :Architecture de LSTM pour classification binaire

1.5.5 Comparaison

Le Tableau 4 fournit une vue synthétique des principales différences entre les quatre modèles de deep learning explorés dans cette étude : MLP, DNN, CNN et LSTM. Chacun de ces modèles repose sur une structure architecturale propre, adaptée à des types de données et à des contextes d'usage spécifiques.

Critère	MLP	DNN	CNN	LSTM
Structure de base	Réseau de neurones entièrement connecté, avec 1 à 2 couches cachées	Extension profonde du MLP avec plusieurs couches cachées	Couches de convolution + pooling + fully connected	Cellules récurrentes avec mémoire à long terme

Type de données	Données tabulaires (vecteurs)	Données tabulaires, structurées	Données spatiales (images, vidéos, signaux 2D)	Données séquentielles (textes, séries temporelles, audio)
Connexion entre neurons	Dense (chaque neurone connecté aux suivants)	Dense sur plusieurs couches	Connexions locales via filtres	Connexions récurrentes dans le temps
Prétraitement requis	Élevé – les caractéristiques doivent être bien préparées	Élevé à moyen	Faible – apprend les caractéristiques spatiales automatiquement	Faible – apprend les relations temporelles
Sensibilité à l'ordre des données	Faible	Faible	Faible	Élevée – prend en compte l'ordre des séquences
Nombre de paramètres à configuré	Modéré	Élevé (augmente avec la profondeur)	Élevé mais optimisé par le partage de poids	Très élevé en raison de la mémoire interne

Table 4: Comparaison entre les modèles (ML) CNN, DNN, MLP, LSTM

Nous observons tout d'abord que le MLP et le DNN sont fondés sur des connexions entièrement denses, avec une complexité croissante selon la profondeur. Le DNN, en tant qu'extension du MLP, est plus adapté aux problèmes nécessitant une capacité d'abstraction élevée, au prix d'un plus grand nombre d'hyperparamètres à configurer.

Le CNN, quant à lui, se distingue par sa capacité à extraire automatiquement des motifs spatiaux à l'aide de filtres de convolution et de couches de pooling, ce qui le rend particulièrement performant pour des données comme les images ou les signaux

2D. Grâce au partage des poids, il offre un bon compromis entre expressivité et efficacité paramétrique, malgré sa profondeur.

En revanche, le LSTM est spécialement conçu pour traiter des données séquentielles : il intègre une mémoire interne capable de retenir des informations sur le long terme, ce qui en fait un choix privilégié dans les cas où l'ordre des données est significatif (ex. : séries temporelles, texte, audio). Toutefois, cette puissance se paie par une complexité de paramétrisation accrue, liée à la gestion des états internes et des connexions récurrentes.

Enfin, du point de vue du prétraitement, les MLP et DNN nécessitent une préparation minutieuse des données, alors que les CNN et LSTM peuvent apprendre directement des représentations pertinentes à partir de données brutes, réduisant ainsi le besoin d'ingénierie de variables.

Cette comparaison souligne que le choix d'un modèle ne repose pas uniquement sur sa performance théorique, mais dépend étroitement de la nature des données manipulées, de la complexité du problème à résoudre et des exigences en matière de réglage. Les hyperparamètres partagés entre les architectures, ainsi que ceux spécifiques à chaque modèle, feront l'objet d'une présentation détaillée dans la suite du chapitre 2.

1.6 Heuristiques et méta heuristiques

La recherche des meilleures configurations d'hyperparamètres (HPO – Hyperparameter Optimization) constitue un problème d'optimisation difficile, souvent intractable par des méthodes exactes. En effet, les approches de résolution exhaustive, telles que la recherche brute (grid search), consistent à évaluer toutes les combinaisons possibles dans l'espace des paramètres. Cette stratégie devient rapidement impraticable dès que la dimension du problème augmente, car elle présente une complexité exponentielle et un coût computationnel prohibitif.

Dans ce contexte, il est souvent préférable de recourir à des méthodes heuristiques, capables de proposer des solutions approchées, mais suffisamment performantes, dans

un temps raisonnable. Ces algorithmes n'ont pas pour objectif de garantir une solution optimale, mais d'explorer efficacement l'espace des solutions. Des techniques comme la recherche aléatoire (RS), la recherche par forêts aléatoires (RF) ou l'optimisation bayésienne figurent parmi les heuristiques les plus couramment utilisées pour l'optimisation d'hyperparamètres.

Toutefois, pour des problèmes plus complexes, des stratégies plus évoluées ont été développées : les métaheuristiques. Contrairement aux heuristiques classiques, les métaheuristiques visent une exploration plus globale et structurée de l'espace des solutions, tout en évitant de se retrouver piégées dans des optima locaux. Leur objectif est de converger progressivement vers un optimum global, sans connaissance a priori de la forme de l'espace de recherche.

Souvent inspirées de phénomènes naturels (biologie, physique, sociologie...), les métaheuristiques sont de plus en plus utilisées en intelligence artificielle et en recherche opérationnelle, et peuvent être hybridées avec d'autres techniques pour en améliorer l'efficacité. Parmi les approches les plus populaires appliquées à la HPO, on retrouve notamment les algorithmes génétiques (AG).

1.6.1 Algorithmes génétiques (AG)

Les algorithmes génétiques font partie de la famille des algorithmes évolutionnaires, qui sont des méthodes stochastiques d'optimisation s'inspirant des principes de la sélection naturelle et de la génétique biologique. Proposés dans les années 1970 par le chercheur américain John Holland, ils s'appuient sur les idées fondatrices de la théorie de l'évolution formulée par Charles Darwin, notamment dans son ouvrage De l'origine des espèces [36].

À l'image de l'évolution des espèces, les AG reposent sur un processus itératif où une population initiale de solutions candidates (appelées individus) évolue génération après génération. À chaque itération, les individus sont évalués à l'aide d'une fonction objectif, puis soumis à des opérateurs génétiques comme :

- la sélection (reproduction des meilleurs individus),
- le croisement (échange d'informations entre solutions),
- et la mutation (modification aléatoire pour introduire de la diversité).

L'ensemble de ce processus suit un cycle composé de cinq phases principales (Figure 10): initialisation, évaluation, sélection, croisement et mutation, suivi d'un retour à l'étape d'évaluation jusqu'à ce qu'un critère d'arrêt soit atteint (nombre d'itérations, convergence, etc.).

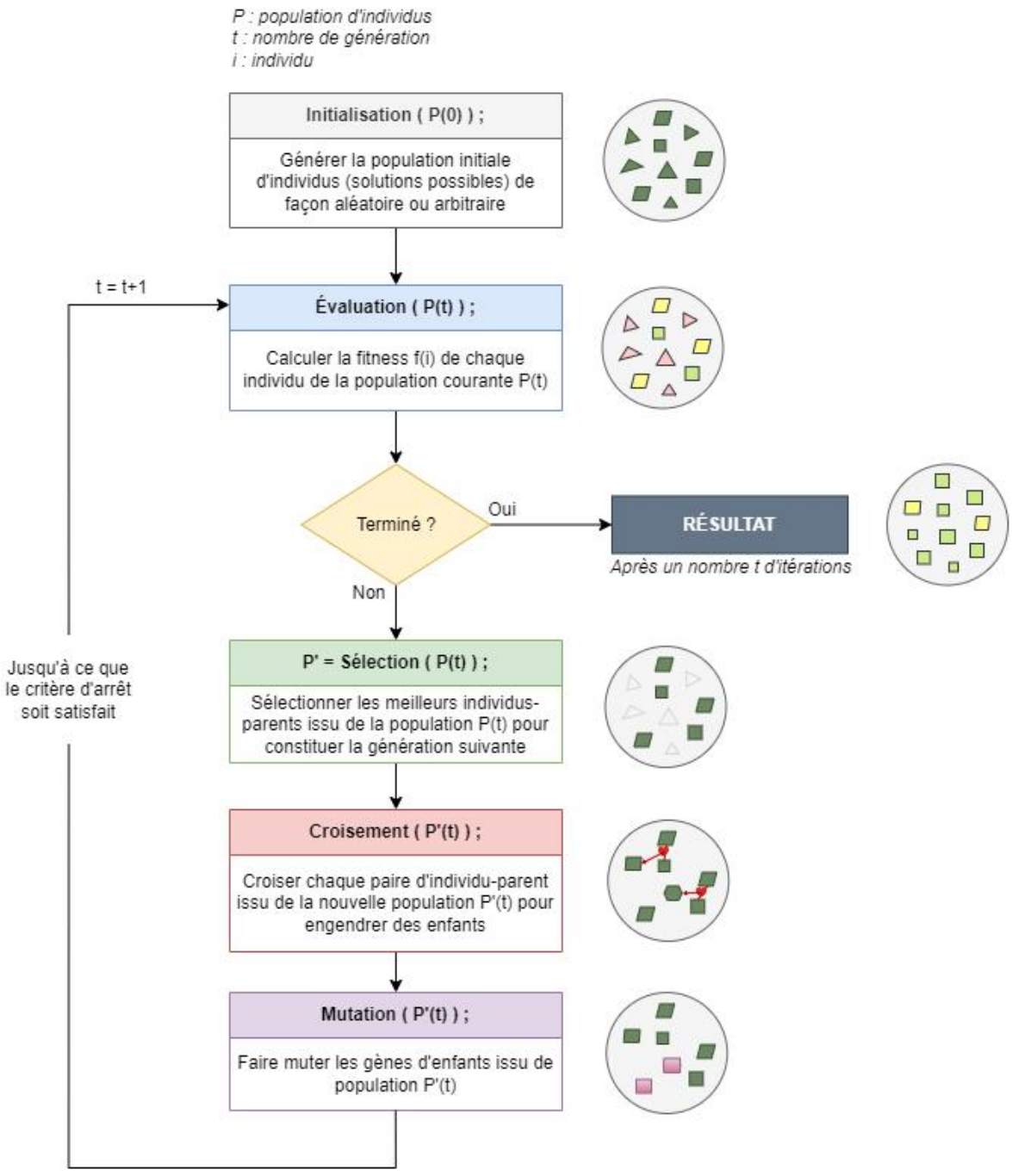

Figure 10 **:** Fonctionnement d'un Algorithme Génétique (GA) simple [37]

Les AG permettent ainsi de faire évoluer une population vers des solutions de plus en plus performantes, tout en conservant un bon équilibre entre exploration (diversité) et exploitation (intensification autour des bonnes solutions). Leur flexibilité et leur capacité à s'adapter à des problèmes complexes font des AG un choix particulièrement pertinent pour résoudre des tâches d'optimisation comme la HPO. Dans le chapitre 2, nous reviendrons en détail sur le fonctionnement algorithmique des AG, leur représentation génétique, ainsi que leur application concrète à l'optimisation des hyperparamètres et à la sélection automatique de modèles.

1.7 Conclusion

Ce chapitre a présenté un état de l'art approfondi sur l'optimisation des hyperparamètres et la sélection en apprentissage automatique, avec un accent particulier sur l'apport des métaheuristiques, notamment dans le contexte des architectures neuronales profondes. L'analyse comparative met en évidence les limites des approches classiques (recherche manuelle, grille, aléatoire) face à la puissance exploratoire des algorithmes génétiques, qui permettent une recherche plus efficace dans des espaces de solutions complexes et de haute dimension. Cette capacité d'adaptation rend ces méthodes particulièrement pertinentes pour les réseaux de neurones profonds.

Ces constats théoriques fondent la démarche expérimentale adoptée dans notre étude, où nous ciblons spécifiquement l'optimisation conjointe des hyperparamètres et la sélection de modèles. Nous appliquons à cet effet l'algorithme génétique (GA) sur quatre architectures majeures du Deep Learning : MLP, DNN, CNN et LSTM. Ces modèles seront décrits et analysés en détail dans le prochain chapitre.

Chapitre II
Contribution

2.1 Introduction

Dans ce chapitre, nous présentons notre contribution à travers deux versions d'un algorithme génétique (AG) adaptées à l'optimisation des modèles d'apprentissage profond (DL). La première version, AG-HPO, se concentre exclusivement sur l'optimisation des hyperparamètres (HPO). La seconde, plus complète, AG-HPO+Select, combine l'optimisation des hyperparamètres avec la sélection du modèle le plus performant parmi une famille de modèles de deep learning. Nous avons appliqué ces deux approches sur quatre architectures courantes en DL : MLP (Multilayer Perceptron), DNN (Deep Neural Network), LSTM (Long Short-Term Memory) et CNN (Convolutional Neural Network).

Pour cela, nous commençons par présenter notre démarche générale, puis nous décrivons brièvement les caractéristiques des modèles considérés. Ensuite, nous procédons à la modélisation formelle des problèmes abordés, à savoir :

- le problème AG-HPO, centré sur la recherche des meilleures combinaisons d'hyperparamètres pour un modèle donné ;
- et le problème AG-HPO+Select, qui étend cette optimisation à la sélection du modèle lui-même.

Nous définissons pour chaque cas la structure d'une solution candidate, ainsi que la fonction objectif à optimiser, avec deux variantes envisagées : une formulation mono-objectif, axée sur le Recall, et une formulation multi-objectif, intégrant à la fois la précision et le rappel à travers le F1-score. Enfin, nous détaillons le fonctionnement des deux versions de l'algorithme génétique, en mettant en lumière leurs différences,

leurs étapes spécifiques, et leur capacité à s'adapter à la nature des modèles considérés. .

2.2 Présentation des architectures de DL utilisées

Dans cette section, nous présentons les quatre architectures de deep learning retenues dans le cadre de cette étude : MLP, DNN, CNN et LSTM. L'analyse commence par une vue d'ensemble de l'architecture générale de ces modèles, suivie d'une description détaillée des spécificités propres à chacun. Une comparaison comparative est ensuite proposée afin de mettre en évidence leurs différences structurelles, fonctionnelles et contextuelles d'application. Une attention particulière est accordée aux hyperparamètres, qu'ils soient communs à tous les modèles ou spécifiques à une architecture donnée.

2.2.1 Vue d'ensemble des architectures DL

L'architecture standard d'un modèle de deep learning pour une tâche de classification binaire se compose généralement de trois blocs principaux : la couche d'entrée (input layer), les couches cachées (hidden layers) et la couche de sortie (output layer). Ce schéma modulaire constitue la base commune des modèles étudiés dans ce travail (MLP, DNN, CNN, LSTM), tout en offrant une grande flexibilité d'adaptation selon le type de données.

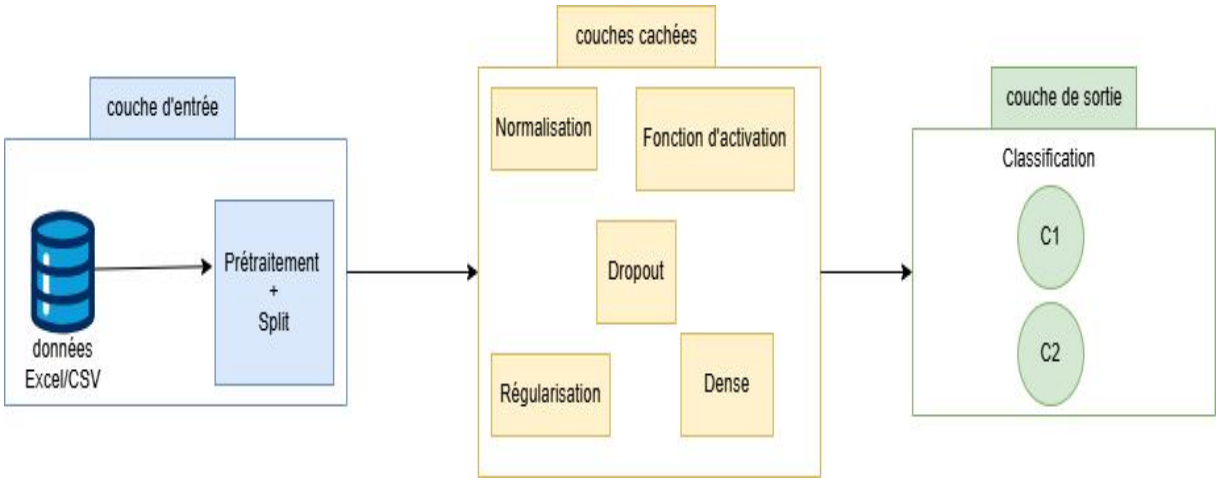

Figure 12: Architecture générale des modèles DL

- Couche d'entrée : Ce premier bloc reçoit les données brutes (images, texte, séries temporelles, tableaux…) et les transforme en représentations numériques exploitables par le réseau. Cette transformation inclut des opérations classiques de prétraitement (nettoyage, normalisation) et le partitionnement (split) des données.
- Couches cachées (noyau du réseau) : C'est au niveau des couches cachées que l'apprentissage proprement dit a lieu. Elles transforment progressivement les caractéristiques d'entrée en représentations de plus en plus abstraites. Ces couches peuvent inclure :
    - Couches denses (Dense / Fully Connected) : Chaque neurone est connecté à tous ceux de la couche précédente.
    - Fonctions d'activation (ReLU, Sigmoid, Tanh) : Introduisent de la non-linéarité pour permettre l'apprentissage de relations complexes.
    - Normalisation (Batch Normalization) : Stabilise l'apprentissage en réduisant la variation des distributions internes.
    - Régularisation (L1, L2) : Pénalise les poids excessifs pour éviter le surapprentissage.
    - Dropout : Désactive aléatoirement certains neurones pendant l'apprentissage pour améliorer la robustesse du modèle.
- Couche de sortie : La couche finale produit la prédiction du modèle. Pour une classification binaire, on utilise soit Pour une classification binaire, on utilise soit un neurone unique avec fonction sigmoïde, retournant une probabilité d'appartenance à une classe, soit deux neurones avec fonction softmax, fournissant une distribution de probabilité sur les deux classes. Ce bloc permet ainsi d'aboutir à une décision finale claire : classe C1 ou C2.

2.2.2 Description individuelle des modèles

Dans cette section, nous détaillons les quatre architectures de deep learning sélectionnées dans le cadre de notre étude. Chaque modèle est présenté en mettant en évidence ses principales caractéristiques structurelles, ses spécificités fonctionnelles ainsi que son adaptabilité aux différentes natures de données (tabulaires, spatiales ou séquentielles). Pour chaque architecture, nous identifions également les hyperparamètres propres, qui influencent directement ses performances, en

complément des paramètres communs évoqués précédemment. Cette analyse comparative permettra de mieux cerner les forces et les limites de chaque modèle dans le contexte de la classification binaire étudiée.

a. MLP (Multilayer Perceptron)

Le Multilayer Perceptron (MLP) représente l'architecture la plus simple parmi les modèles étudiés. Il est particulièrement adapté aux données tabulaires. Ce modèle repose exclusivement sur un empilement de couches entièrement connectées (dense layers), sans intégration de structure spatiale ou temporelle. Les données d'entrée, normalisées sont directement injectées dans la première couche du réseau. La structure du MLP peut inclure plusieurs couches cachées, chacune composée d'une couche dense, éventuellement suivie d'une normalisation et d'un dropout pour améliorer la robustesse du modèle. Ce motif simple est répété pour permettre une transformation progressive des données.

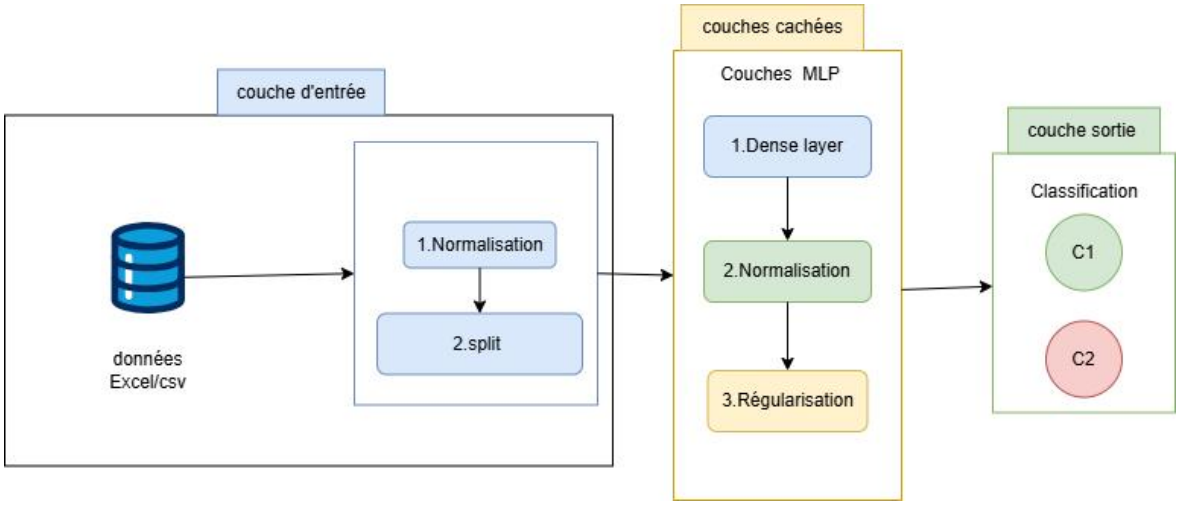

Figure 13: Architecture de *modèle* MLP

b. DNN (Deep Neural Network)

Le Deep Neural Network (DNN) reprend les principes fondamentaux du MLP, mais avec une profondeur accrue. Il s'agit d'une architecture dense à plusieurs niveaux, conçue pour capturer des relations complexes entre les variables d'entrée, particulièrement adaptée aux jeux de données tabulaires riches et non linéaires. Le processus débute par une étape de prétraitement systématique, où les données tabulaires sont normalisées afin d'uniformiser les échelles des variables et favoriser la convergence. Le cœur du modèle est constitué d'une succession de blocs denses profonds, dont le nombre et la configuration sont dynamiquement ajustables. Chaque bloc comprend :

- Une couche dense, capable de modéliser des interactions non linéaires complexes ;
- Une couche de normalisation (Batch Normalization) pour stabiliser l'optimisation dans les architectures profondes ;
- Des mécanismes de régularisation tels que le dropout ou les pénalités L1/L2, intégrés pour limiter le surapprentissage.

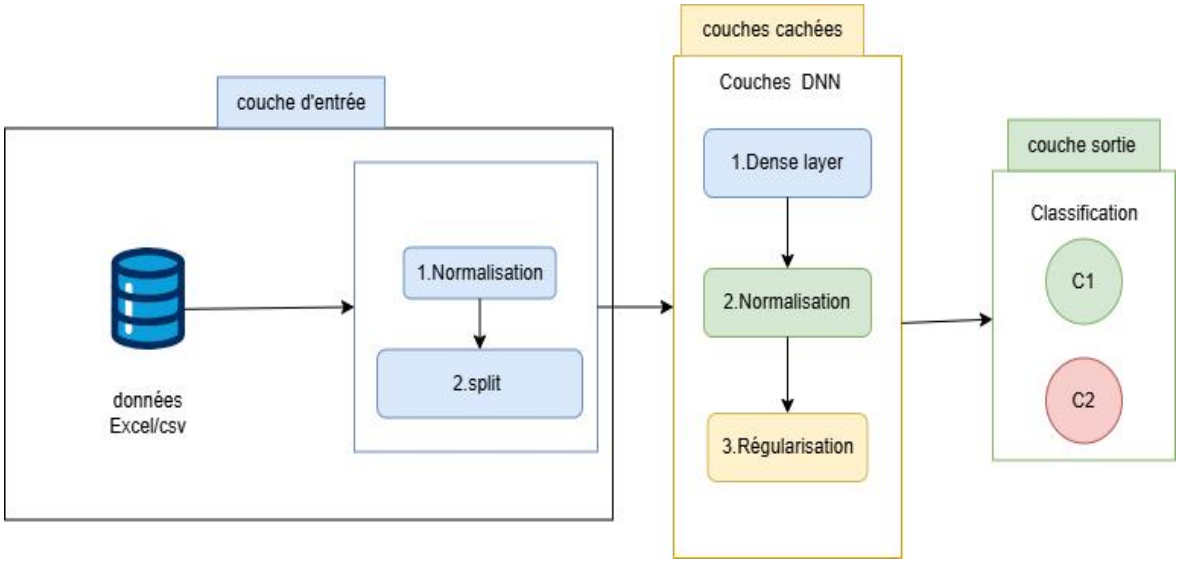

Figure 14 : Architecture de modèle DNN

c. LSTM (Long Short-Term Memory)

Le modèle LSTM (Long Short-Term Memory), issu de la famille des réseaux neuronaux récurrents (RNN), est initialement conçu pour traiter des données séquentielles et capturer des dépendances temporelles à long terme. Après une

normalisation, ce modèle est adapté à des données tabulaires en effectuant un reshape artificiel pour les convertir au format attendu par la couche LSTM : (samples, 1, features). Cette transformation impose une lecture séquentielle unitaire des caractéristiques, bien que les données d'origine ne possèdent pas de structure temporelle intrinsèque. Par conséquent, la LSTM agit ici comme une couche dense enrichie, sans exploiter son potentiel dynamique. Le cœur du modèle repose sur une ou plusieurs couches LSTM, qui traitent les observations reformées comme de fausses séquences. Cette structure est ensuite complétée par une séquence classique de couches de régularisation (Dropout, pénalisation L1/L2), de normalisation (Batch Normalization), et de couches denses fully connected pour finaliser l'apprentissage discriminatif.

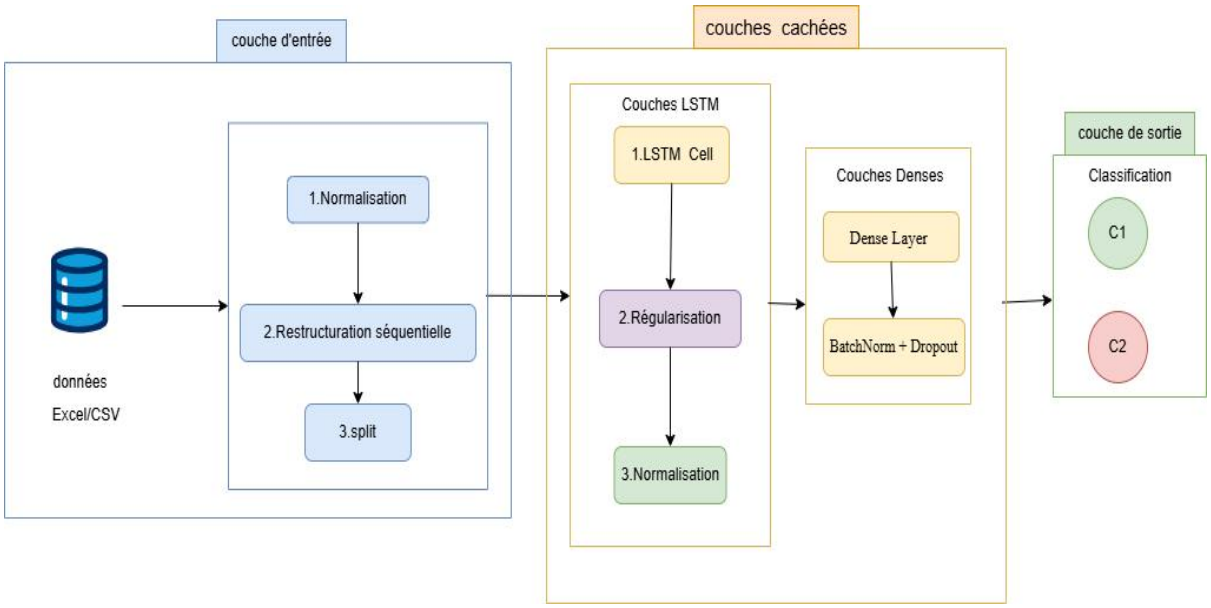

Figure 15: Architecture de *modèle* LSTM

d. Modèle CNN :

Le Convolutional Neural Network (CNN) est une architecture conçue à l'origine pour le traitement de données à structure spatiale, telles que les images ou les signaux multidimensionnels. Dans notre contexte, bien que les données soient tabulaires, le CNN est utilisé pour extraire automatiquement des motifs locaux à travers des filtres convolutifs appliqués sur les vecteurs d'entrée transformés.

Les données, préalablement normalisées, subissent une transformation en pseudo-images 1D via un reshape artificiel, leur donnant une forme compatible avec les couches de convolution. Ce format permet d'appliquer des filtres glissants (kernels) capables de détecter des régularités locales, même dans des données non visuellement structurées.

- L'architecture CNN repose sur une séquence de blocs composés :
- d'une couche de convolution 1D, avec un certain nombre de filtres et une taille de noyau à définir ;
- d'une couche de pooling (ex. : max pooling) pour réduire la dimension tout en conservant les caractéristiques essentielles ;
- de couches de normalisation (Batch Normalization) et de dropout pour régulariser l'apprentissage ;
- et enfin de couches denses fully connected pour la phase de classification.

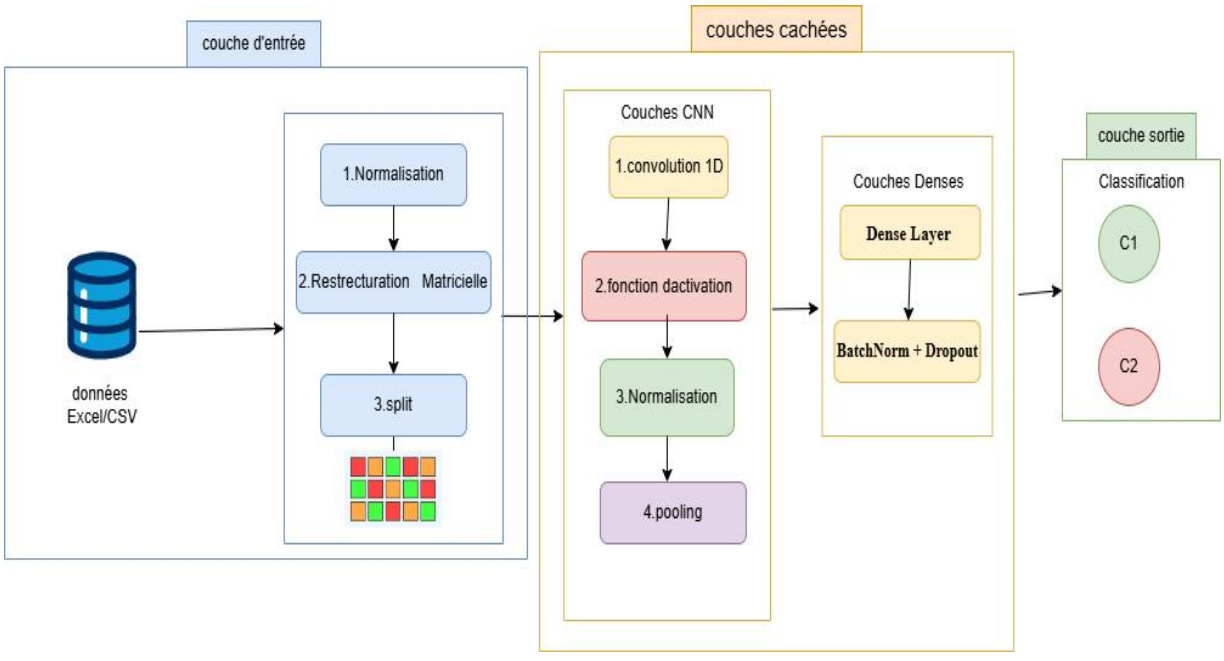

Figure 16 : Architecture de CNN

2.2.3 Analyse comparative des quatre architectures

Cette section propose une analyse comparative structurée des quatre modèles de deep learning retenus dans notre étude : MLP, DNN, CNN et LSTM. L'objectif est de caractériser chaque architecture selon des critères techniques clés, en mettant en évidence leurs spécificités, leurs exigences en matière de données d'entrée, leur complexité, ainsi que leurs avantages respectifs.

Critère	CNN	LSTM	DNN	MLP
Type de couches principales	Conv1D + Pooling	Couches LSTM	Couches denses	Couches fully-connected
Architecture	Complexe	Séquentielle	Directe	Classique
Spécificité	Détection de patterns (local patterns)	Traitement séquentiel avec mémoire	Apprentissage général dense	Dense avec structure optimisée
Dropout	Standard (optionnel)	Dropout récurrent	Standard (optionnel)	Standard (optionnel)
Format d'entrée requis	3D : (samples, features, 1)	3D temporel : (samples, 1, features)	2D : (samples, features)	2D : (samples, features)
Utilisation typique	Analyse de séquences / signaux	Séries temporelles / séquences	Données tabulaires généralistes	Cas simples, baseline efficace
Niveau de complexité	Élevé	Moyen	Moyen à faible	Faible
Avantage clé	Extraction hiérarchique des motifs	Mémorisation de dépendances temporelles	Simplicité et rapidité	Facile à entraîner et à déployer

Table 6: Comparaison descriptive des architectures neuronales CNN, LSTM, DNN et MLP

Ce tableau ne vise pas à anticiper les performances empiriques, mais constitue un élément fondamental de définition de notre stratégie de recherche. Il remplit trois fonctions essentielles dans notre démarche scientifique :

- Délimiter l'espace des solutions : Il justifie le choix des quatre architectures en montrant qu'elles couvrent un large spectre de paradigmes d'apprentissage :
    - CNN pour la détection automatique de motifs locaux,
    - LSTM pour la gestion de séquences et de dépendances temporelles,
    - DNN pour l'apprentissage dense complexe,
    - MLP comme référence simple et généralisée.
- Modéliser la complexité du problème : Chaque architecture implique un ensemble distinct d'hyperparamètres à ajuster. Par exemple, un CNN peut nécessiter jusqu'à 18 hyperparamètres (comme nous allons voir ultérieurement), ce qui augmente significativement la dimension de l'espace de recherche. Ce constat renforce la pertinence de notre approche, qui repose sur des techniques d'optimisation avancées — en particulier, les algorithmes génétiques.
- Préparer le cadre expérimental : Ce tableau introduit la problématique centrale des expérimentations : « Parmi ces architectures théoriquement pertinentes, laquelle offrira la meilleure performance une fois optimisée ? »

2.3 Conclusion

Ce chapitre a présenté en détail la mise en œuvre de notre approche d'optimisation basée sur les algorithmes génétiques, appliquée aux architectures de réseaux neuronaux profonds. Deux stratégies ont été explorées : une optimisation individuelle des hyperparamètres (HPO) propre à chaque modèle, et une optimisation conjointe combinant sélection de l'architecture et réglage des paramètres (HPO + Sélection). Ces deux variantes ont été déployées avec succès sur les quatre architectures étudiées : MLP, DNN, CNN et LSTM.

Le recours à des fonctions de fitness adaptées — notamment le Recall, priorisé pour la sensibilité en détection, et le F1-Score, pour un compromis entre précision et rappel — a permis d'orienter efficacement la recherche vers des configurations performantes et robustes.

L'ensemble de cette modélisation constitue le socle de l'analyse expérimentale présentée dans le chapitre suivant, où les résultats obtenus seront évalués, comparés et interprétés.

Chapitre III :

Implémentation ET Résultats

3.1 Introduction

Ce chapitre est consacré à la mise en œuvre pratique de notre approche ainsi qu'à l'évaluation expérimentale des résultats obtenus. Nous y décrivons dans un premier temps les outils et technologies utilisés pour l'implémentation. Nous présentons ensuite l'implémentation de l'algorithme génétique (GA) développé dans le cadre de ce travail. La troisième partie est dédiée à la description de l'environnement de test, incluant le jeu de donnée utilisé, ainsi que les paramètres expérimentaux adoptés. Enfin, nous exposons les résultats obtenus, accompagnés d'une analyse comparative et synthétique, permettant d'évaluer l'efficacité de la méthode proposée.

3.2 Outils d'*implémentation*

L'implémentation de notre système d'optimisation pour l'hyperparamétrage et la sélection de modèles en machine learning repose sur l'intégration cohérente d'outils spécialisés couvrant l'ensemble du processus de classification automatique, depuis le

prétraitement des données jusqu'à l'évaluation finale. La figure 1 illustre cette chaîne technologique, et le tableau ci-dessous en résume les composants :

- Prétraitement des données : pandas et NumPy pour la manipulation, le nettoyage et la transformation des données.
- Entraînement des modèles : TensorFlow et Keras pour la construction, l'entraînement et l'évaluation des modèles de deep learning (MLP, DNN, CNN, LSTM, etc.).
- Optimisation des hyperparamètres (HPO): scikit-learn pour les stratégies de recherche aléatoire (RandomizedSearchCV) et de recherche en grille (GridSearchCV), S**cikit-optimize (skopt)** pour l'optimisation bayésienne efficace et automatisée et DEAP pour la mise en œuvre de l'algorithme génétique appliqué à la recherche des hyperparamètres optimaux.
- Évaluation des performances : scikit-learn : pour le calcul des métriques (précision, rappel, F1-score, etc.), la validation croisée et la sélection de modèles.

Cette combinaison d'outils offre un environnement flexible et puissant pour explorer, entraîner, optimiser et évaluer les modèles de manière rigoureuse.

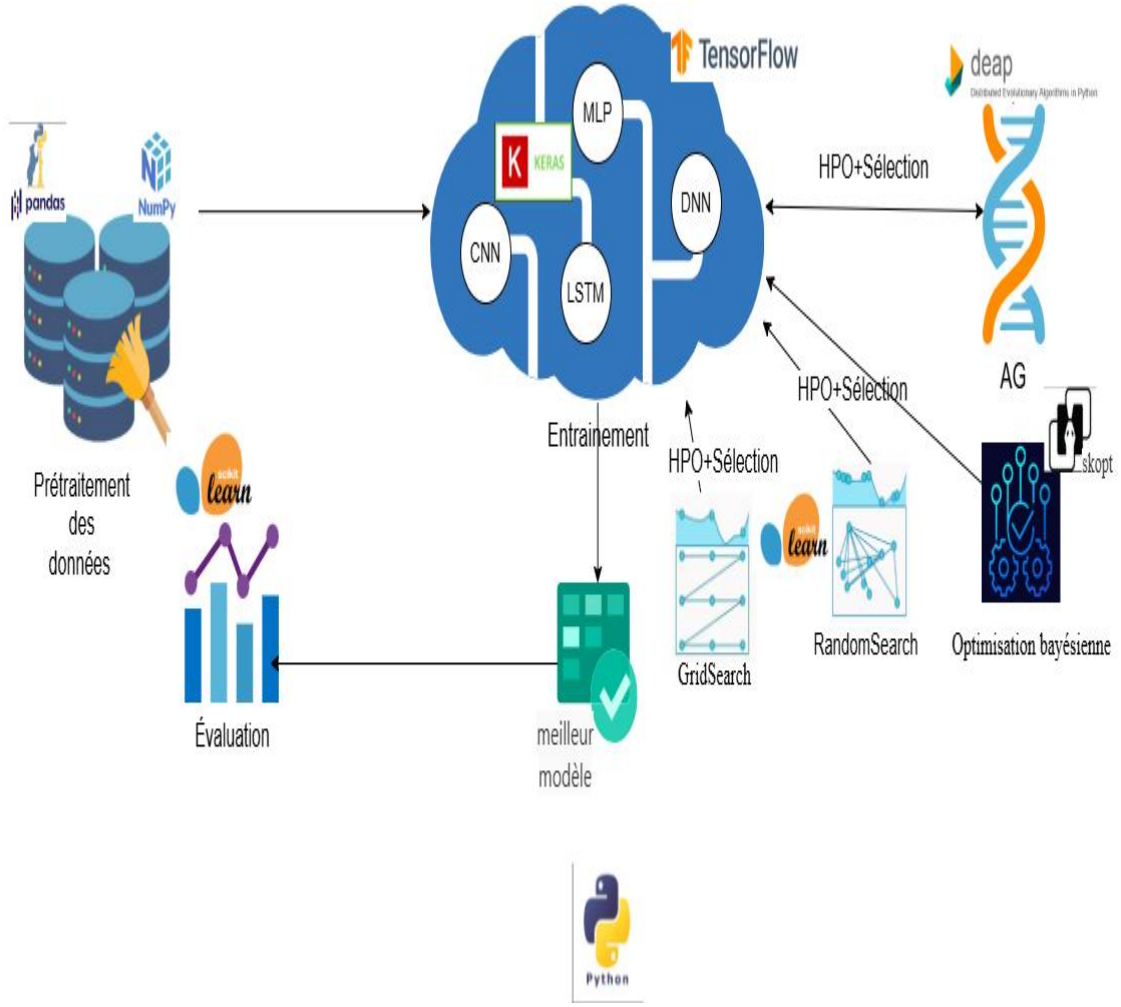

Figure 21: Environnement logiciel

Outil	Description
Python	Un langage de programmation interprété, de haut niveau et orienté objet, connu pour sa syntaxe claire et sa facilité d'apprentissage. Il dispose de structures de données avancées et d'un écosystème riche de bibliothèques, ce qui facilite le développement rapide de scripts et d'applications dans de nombreux domaines. Python est largement utilisé en data science et IA grâce à sa polyvalence et à ses nombreuses bibliothèques dédiées [18]
TensorFlow	Une plateforme open source développée par Google pour l'apprentissage automatique (machine learning) et le deep learning. C'est « une plate-forme de bout en bout dédiée au machine learning », offrant un écosystème complet d'outils et de bibliothèques pour construire,

	entraîner et déployer des modèles de réseaux de neurones dans divers environnements. TensorFlow facilite la création de modèles ML complexes et leur exécution efficace sur différentes plateformes grâce à son API haut niveau et ses optimisations sous-jacentes.[19]
Keras	Une bibliothèque Python open source de haut niveau pour le deep learning (apprentissage profond). Elle fournit une interface simple et intuitive pour créer et entraîner des modèles de réseaux de neurones, tout en s'appuyant sur des frameworks bas-niveau comme TensorFlow (ou d'autres backends) pour exécuter les calculs. En d'autres termes, Keras est une « surcouche » à TensorFlow qui simplifie le prototypage rapide de modèles complexes grâce à des blocs modulaires réutilisables.[20]
scikit-learn	Une bibliothèque Python open source d'apprentissage automatique (machine learning) généraliste. Elle offre un large ensemble d'algorithmes et d'outils pour l'apprentissage supervisé (classification, régression) et non supervisé, ainsi que pour l'évaluation et la validation de modèles. Scikit-learn est conçue pour rendre les tâches de modélisation statistiques et d'exploration de données accessibles et efficaces en Python.[21]
*S*cikit-optimize	est une bibliothèque Python open source conçue pour l'optimisation efficace des hyperparamètres basée sur des méthodes bayésiennes. Elle s'intègre parfaitement avec les bibliothèques comme Scikit-Learn, et vise à rendre l'optimisation plus intelligente, rapide et automatique.[22]
DEAP	Un framework Python pour le calcul évolutionnaire et l'optimisation par algorithmes génétiques ou évolutionnaires. Il permet de prototyper rapidement de nouvelles méthodes d'optimisation et d'intelligence artificielle évolutionnaire grâce à des opérateurs génériques (mutation, croisement, sélection) et prend en charge le parallélisme pour accélérer les calculs. DEAP facilite ainsi la conception et le test d'algorithmes évolutionnaires complexes en rendant explicites les étapes des algorithmes et en optimisant la représentation des données.[23]

Pandas	Une bibliothèque Python open source dédiée à l'analyse et à la manipulation de données. Elle introduit des structures de données flexibles, notamment le *DataFrame* , qui simplifient le traitement des données tabulaires. Pandas fournit des fonctions performantes pour filtrer, agréger, nettoyer et transformer les données, et constitue l'outil standard en Python pour la préparation et l'analyse de grands ensembles de données.[24]
NumPy	La bibliothèque fondamentale du calcul scientifique en Python. Elle fournit le type de données *ndarray*, un tableau multidimensionnel homogène, ainsi qu'un grand nombre de routines vectorisées pour effectuer efficacement des opérations mathématiques, logiques et statistiques sur ces tableaux. NumPy permet d'accélérer les calculs sur de grands ensembles de données en exécutant les opérations en code compilé, ce qui en fait la base de nombreux autres outils (pandas, scikit-learn, TensorFlow, etc.) dans l'écosystème Python pour la data science.[25]

Table 11::Description des Outils d 'implémentation

3.3 Environnement Expérimental

Les expérimentations ont été menées sur deux environnements distincts : un poste local utilisé principalement pour le développement et le débogage, et un environnement cloud (Kaggle Notebooks) dédié à l'exécution finale des algorithmes et à l'optimisation génétique. Le tableau ci-dessous résume les configurations matérielles et logicielles associées à chacun de ces environnements :

Composant	Poste local (Développement)	Kaggle Notebooks (Exécution principale)
Nom du système	DESKTOP-AEIJ8HD	Serveur Kaggle (Cloud)
Processeur	Intel Core i7-8650U (4 cœurs/8 threads)	4 cœurs Intel Xeon @ 2.20 GHz
Vitesse d'horloge	1.90 GHz → 2.11 GHz (Turbo Boost)	Non spécifié
GPU	CPU uniquement	2× NVIDIA Tesla T4 (15 Go mémoire/GPU)
Capacité GPU	/	8.1 TFLOPS (FP32) par carte
RAM	8 Go DDR4	29 Go
Stockage	SSD NVMe	Stockage cloud Kaggle
Système d'exploitation	Windows 10 Pro 64-bit (Build 19045)	Linux (Debian 12)
Python	3.10	3.10
TensorFlow	2.10 (Mode CPU)	2.18.1 (CUDA 12.x)
Bibliothèques clés	NumPy, Pandas, Scikit-learn, DEAP, Keras	Scikit-learn 1.2.2, DEAP 1.4.3

Table 12:Evironnement matériel et logiciel

3.3.1 JEU DE DONNEE UTILISEE

Ce projet s'inscrit dans une démarche visant à développer des modèles d'apprentissage automatique plus performants, robustes et adaptés aux réalités complexes des données, notamment dans des domaines critiques tels que la cybersécurité. L'un des défis majeurs abordés est la détection des ransomwares, des logiciels malveillants qui chiffrent les données de leurs victimes en échange d'une rançon. Cette menace, en constante évolution, exige des solutions d'apprentissage capables de distinguer efficacement les comportements malveillants dans un environnement de plus en plus complexe.

Pour évaluer notre approche, nous avons sélectionné le Ransomware PE Header Feature Dataset proposé par Moreira et al. (2023a). Ce jeu de données se distingue par sa diversité (avec 25 familles de ransomwares), sa répartition équilibrée entre les classes (goodware et ransomware), et sa crédibilité académique, étant régulièrement utilisé dans des travaux de recherche en cybersécurité. Il constitue ainsi une base idéale pour tester la capacité des modèles à gérer des données hétérogènes et à détecter efficacement les menaces dans un contexte réaliste.

La structure du dataset est résumée dans le tableau suivant :

Caractéristique	**Valeur**
Taille totale	2 157 échantillons
Nombre de goodwares	1 134
Nombre de ransomwares	1 023
Nombre de familles	25 familles de ransomwares
Type de données	Tabulaires (extraits des en-têtes PE)

Répartition des classes	Relativement équilibrée
Domaine d'application	Cybersécurité – détection de malwares
Adaptabilité aux modèles	Directement compatible avec MLP/DNN
Traitement requis pour CNN/LSTM	Restructuration des vecteurs tabulaires en formats matriciels ou séquentiels compatibles

Table 13:Structure du jeu de données

Étant donné que ce dataset est tabulaire, son utilisation directe est adaptée aux modèles classiques comme les MLP ou DNN. En revanche, pour exploiter des architectures plus avancées comme les CNN et LSTM, une transformation préalable des données est nécessaire. Cette étape consiste à restructurer les vecteurs plats en formats matriciels ou séquentiels compatibles avec ces réseaux. Cette adaptation permet d'explorer la capacité des modèles convolutifs et récurrents à extraire des motifs temporels ou spatiaux même à partir de données tabulaires initialement statiques.

3.3.2 Protocole expérimental

Ce protocole a été mis en place pour évaluer l'efficacité de notre méthode d'optimisation des hyperparamètres appliquée à différents modèles de deep learning pour la détection de ransomwares. Il se déroule en trois étapes principales :

Étape 1 : Optimisation des hyperparamètres pour chaque modèle

Dans un premier temps, nous avons appliqué l'algorithme génétique séparément à chaque architecture (MLP, DNN, CNN, LSTM). L'objectif était d'optimiser les hyperparamètres propres à chaque modèle sans comparaison entre eux. Les performances ont été mesurées à l'aide des métriques standard : **Accuracy**, **Recall**, **F1-score** et **Précision**. Cette étape permet d'évaluer le potentiel de chaque modèle une fois bien réglé.

Étape 2 : Optimisation combinée des hyperparamètres et de l'architecture

Ensuite, nous avons élargi l'approche pour que l'algorithme génétique optimise à la fois les hyperparamètres et le choix du modèle. Chaque solution candidate représente une combinaison complète (modèle + configuration). Le but est de trouver automatiquement la meilleure architecture avec ses paramètres optimaux selon deux critères principaux : **Recall** pour maximiser la détection, et **F1-score** pour garantir un bon équilibre global.

Étape 3 : Évaluation comparative des techniques d'optimisation

Enfin, notre stratégie évolutive a été confrontée à trois méthodes alternatives d'optimisation des hyperparamètres : deux méthodes classiques, à savoir **Grid Search** et **Random Search**, ainsi qu'une méthode automatisée, l'**optimisation bayésienne**. La comparaison a porté sur trois axes : la qualité finale du modèle (architecture + paramètres), les performances obtenues en Recall et F1-score, et le coût computationnel (temps d'exécution) nécessaire pour atteindre ces résultats.

Ce protocole structuré nous a permis de tester la robustesse et l'efficacité de notre approche dans un cadre expérimental contrôlé, tout en la situant par rapport aux techniques de référence couramment utilisées en machine learning.

3.3.3 Paramètres de l'expérimentation

Afin de garantir la reproductibilité et la rigueur méthodologique de notre protocole expérimental, l'ensemble des paramètres liés à notre approche d'optimisation des hyperparamètres et de sélection des modèles de deep learning ont été définis et détaillés dans les sections précédentes. Ceux-ci comprennent notamment :

- les réglages de l'algorithme génétique (taille de population, nombre de générations, probabilités de croisement et de mutation) [cf. Chapitre 2, Section Algorithme GA-HPO],

- les options d’encodage des individus (type de modèle et hyperparamètres spécifiques) [cf. Chapitre 2, Section Hyperparamètres de nos modèles],
- les paramètres de validation (stratégie de partitionnement, proportion de données de validation) [cf. Chapitre 2, Section Notre approche],
- les critères d’évaluation utilisés pour mesurer la performance (Recall, F1-score, etc.) [cf. Chapitre 2, Section Notre approche],

ainsi que les spécifications de l’environnement matériel utilisé pour l’exécution des expériences [cf. Section outils d’implémentation].

3.4 CONCLUSION :

Ce chapitre a présenté l'ensemble du protocole expérimental mis en œuvre pour évaluer notre approche d'optimisation des hyperparamètres et de sélection automatique de modèles appliquée à la détection des ransomwares. Après avoir détaillé les outils technologiques utilisés, l'implémentation de l'algorithme génétique, ainsi que les caractéristiques du dataset et de l'environnement de test, nous avons procédé à une double phase d'évaluation : individuelle (par architecture) et collective (multi-modèles).

Les résultats obtenus montrent que notre méthode permet d'améliorer significativement les performances des modèles, notamment en maximisant les métriques critiques que sont le Recall et le F1-score, tout en maintenant une bonne précision et une exécution maîtrisée. Le modèle MLP, identifié comme le meilleur candidat dans l'approche multi-modèle, illustre la capacité de notre algorithme à sélectionner automatiquement la combinaison optimale modèle + hyperparamètres, sans intervention humaine.

Enfin, les comparaisons menées avec des méthodes classiques d'optimisation (Grid Search, Random Search, Optimisation bayésienne) confirment l'efficacité et la robustesse de notre approche à base d'algorithme génétique, qui offre un bon compromis entre qualité des résultats et coût computationnel.

Ce chapitre constitue ainsi une validation expérimentale solide de notre contribution, en posant les bases pour des extensions futures vers des cas d'usage plus larges ou des environnements en production.

CONCLUSION GÉNÉRALE et Perspectives

Ce mémoire s'inscrit dans une démarche d'amélioration des performances en apprentissage automatique, en ciblant à la fois l'optimisation des hyperparamètres et la sélection automatique du modèle. L'objectif a été de démontrer l'apport de l'algorithme génétique (GA) pour identifier, de manière conjointe, les meilleures combinaisons de paramètres et l'architecture de deep learning la plus adaptée parmi MLP, DNN, CNN et LSTM. L'approche développée vise à dépasser les limites des méthodes classiques telles que Grid Search, Random Search ou l'optimisation bayésienne, en offrant une solution plus efficace en termes de précision, de rapidité de convergence et de maîtrise du coût computationnel.

Pour ce faire, ce mémoire a d'abord établi un état de l'art sur les modèles neuronaux profonds, les techniques d'optimisation des hyperparamètres, ainsi que les métaheuristiques, avec un accent particulier sur les algorithmes génétiques. Deux approches ont été proposées :

- GA-HPO : une version mono-modèle visant l'optimisation spécifique des hyperparamètres d'une seule architecture ;

- GA-HPO+Select : une version enrichie, combinant optimisation des hyperparamètres et sélection automatique du modèle le plus adapté.

Ces deux approches ont été appliquées à quatre architectures de deep learning largement utilisées (MLP, DNN, CNN, LSTM) dans un contexte de détection de

ransomwares, en utilisant des fonctions de fitness adaptées : le Recall pour une optimisation mono-objectif, et le F1-score pour une optimisation multi-objectifs.

Les résultats expérimentaux obtenus confirment l'efficacité de notre méthode. Les modèles optimisés par l'algorithme génétique ont surpassé les solutions issues des méthodes classiques tant en termes de performances que de stabilité. Notamment, l'approche GA-HPO+Select a permis d'identifier automatiquement le modèle MLP comme étant le plus performant dans notre cas d'étude, démontrant la capacité du GA à explorer intelligemment l'espace des solutions tout en automatisant le processus de conception.

Plusieurs pistes peuvent prolonger ce travail :

- Étendre l'approche à d'autres types de modèles (transformers, auto-encodeurs, etc.) ou à des architectures hybrides.

- Intégrer des contraintes supplémentaires dans l'optimisation, telles que la consommation mémoire ou le temps d'inférence, dans une perspective de déploiement temps réel.

- Explorer d'autres métaheuristiques ou approches hybrides combinant GA avec des stratégies bayésiennes ou de méta-apprentissage.

- Enfin, appliquer cette méthodologie à d'autres domaines applicatifs (santé, finance, industrie) pour tester sa robustesse sur des cas d'usage variés.

Ainsi, cette étude démontre non seulement la pertinence des algorithmes génétiques pour l'optimisation conjointe modèle–hyperparamètres, mais elle ouvre également la voie à des recherches plus poussées vers des systèmes intelligents et totalement autonomes d'optimisation en machine learning..

BIBLIOGRAPHIE

[1] Shaveta (2023). À Review on Machine Learning. International Journal of Science and Research Archive, 9(1), 281–285.

[2] GeeksforGeeks (2025, 27 mai). Supervised vs Unsupervised vs Reinforcement Learning.

[3] Lifewire (2023). *What Is* Supervised Learning ?

[4] Vishnu Vardhan Baligodugula & Fathi Amsaad (2025). Unsupervised Learning: Comparative Analysis of Clustering Techniques on High-Dimensional Data

[5] Tang, C., Abbatematteo, B., Hu, J., Chandra, R., Martín-Martín, R., & Stone, P. (2025). *Deep Reinforcement Learning for Robotics: A Survey of Real-World Successes*. *Annual Review of Control, Robotics, and Autonomous Systems*, 8, 153-188.

[6] Sharafat Ahmed (2025, juin). The Supervised Learning Process in Machine Learning. Medium.

[7] M. L. Madhan (2023, 6 août). *Ultimate Guide to Machine Learning Pipeline*. *Medium*.

[8] Kıvanç, Ş. G., Şen, B., Nar, F., & Ok, A. Ö. (2024). *Reducing Model Complexity in Neural Networks by Using Pyramid Training Approaches*. *Applied Sciences*, 14(13), 5898.

https://doi.org/10.3390/app14135898

[9] Súkeník, P., & Lampert, C. (2024). *Generalization in multi-objective machine learning*. *Neural Computing & Applications*.

https://doi.org/10.1007/s00521-024-10616-

[10] Ilemobayo, J. A., Durodola, O., Alade, O., & Awotunde, O. J. (2024). Hyperparameter Tuning in Machine Learning: A Comprehensive Review. Journal of Engineering Research and Reports, 26(6), 388–395.

https://doi.org/10.9734/jerr/2024/v26i61188

[11] Liu, H., Sim, K. C., & Cao, B. (2023). *Hyperparameter Optimization Techniques in Machine Learning: A Comprehensive Review and Future Directions.* IEEE Transactions on Neural Networks and Learning Systems, 34(7), 2892-2910. https://doi.org/10.1109/TNNLS.2022.3155892

[12] L. Feurer, F. Hutter, *"Hyperparameter Optimization"*, in Automated Machine Learning: Methods, Systems, Challenges, edited by F. Hutter, L. Kotthoff, J. Vanschoren, Springer, 2019, pp. 3-33.

[13] Y. LeCun, Y. Bengio, G. Hinton, *Deep learning*, Nature, vol. 521, no. 7553, pp. 436-444, 2015.
DOI : 10.1038/nature14539

[14] Géron, A. (2023). Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow (3rd ed.). O'Reilly Media

[15] Kocbek, R., Stare, J., & Rozman, A. (2024). Multilayer Perceptron: Architecture Optimization for Classifying Anemia Patients. In Proceedings of AI Technologies for Information Systems and Management Science

[17]https://www.ibm.com/fr-fr/think/topics/confusion matrix#:~:text=La%20matrice%20de%20confusion%20permet,d'un%20jeu%20de%20donn%C3%A9es.

[18] Van Rossum, G., & Drake, F. L. (2009). *Python 3 Reference Manual*. Python Software Foundation.

[19] Abadi, M. et al. (2016). *TensorFlow: A System for Large-Scale Machine Learning*. OSDI.

[20] Chollet, F. (2015). *Keras*. https://keras.io

[21] Pedregosa, F. et al. (2011). *Scikit-learn: Machine Learning in Python*. JMLR.

[22] Head, T. et al. (2018). *Scikit-Optimize: Sequential model-based optimization with a* scikit-learn *interface*. https://scikit-optimize.github.io

[23] Fortin, F.-A., et al. (2012). *DEAP: Evolutionary Algorithms Made Easy*. JMLR.

[24] McKinney, W. (2010). *Data Structures for Statistical Computing in Python*. Proceedings of the 9th Python in Science Conference.

[25] Harris, C. R., et al. (2020). *Array programming with NumPy*. Nature.

[26] Amalou, N., & Souaci, F. Z. (2024). *Ransomware Detection using Deep Learning* [Master's thesis, Blida 1 University]. Computer Science Department, Faculty of Sciences.

[27] Ouedraogo, I. (*s.d.*).
Automatisation du pré-traitement des données par l'optimisation métaheuristique (Mémoire de maîtrise, Université du Québec à Chicoutimi, Canada).

[28] Laredo, D., Qin, Y., Schütze, O., & Sun, J.-Q. (*s.d.*).
Automatic Model Selection for Neural Networks.
University of California, Merced & CINVESTAV, Mexico.

[29] Sen, A., Mazumder, A. R., Dutta, D., Sen, U., Syam, P., & Dhar, S. (*s.d.*).
Comparative Evaluation of Metaheuristic Algorithms for Hyperparameter Selection in Short-Term Weather Forecasting.
Indian Institute of Engineering Science and Technology, Carnegie Mellon University, NIT Durgapur.

[30] Hussain, W., Mushtaq, M. F., Shahroz, M., Akram, U., Ghith, E. S., Tlija, M., Kim, T.-H., & Ashraf, I. (*s.d.*).
Ensemble Genetic and CNN Model-Based Image Classification by Enhancing Hyperparameter Tuning.

[31] Mohakud, R., & Dash, R. (2022).
Designing a Grey Wolf Optimization Based Hyper-Parameter Optimized Convolutional Neural Network Classifier for Skin Cancer Detection.
Journal of King Saud University – Computer and Information Sciences, 34, 6280–6291.

[32] Purnomo, H. D., Gonsalves, T., Mailoa, E., Santoso, F. Y., & Pribadi, M. R. (2024).
Metaheuristics Approach for Hyperparameter Tuning of Convolutional Neural Network.
JURNAL RESTI, 8(3), 340–345.
[e-ISSN: 2580-0760] Publié en ligne : http://jurnal.iaii.or.id

[33] Google Developers. (2025, 22 mai). *Classification: Accuracy, recall, precision, and related metrics*. Dans *Machine Learning Crash Course*. Récupéré de Google Developers

[34] Bergstra, J., & Bengio, Y. (2012). *Random Search for Hyper-Parameter Optimization*. Journal of Machine Learning Research, 13, 281–305

[35] Snoek, J., Larochelle, H., & Adams, R. P. (2012). *Practical Bayesian Optimization of Machine Learning Algorithms*. Advances in Neural Information Processing Systems, 25, 2951–2959

[36] Alam, T., Qamar, S., Dixit, A., & Benaida, M. (2020). *Genetic Algorithm: Reviews, Implementations, and Applications*. *International Journal of Engineering Pedagogy (iJEP), 10*(6), 57–77.

[37] Albadr, M., Tiun, S., Ayob, M., & Al-Dhief, F. T. (2020). *Genetic Algorithm Based on Natural Selection Theory for Optimization Problems* [Figure 1: Flowchart of the standard genetic algorithm (GA)]. *ResearchGate*.