الجمهورية الجزائرية الديمقراطية الشعبية
République Algérienne Démocratique et Populaire
وزارة التعليم العالي والبحث العلمي

Ministère de l'Enseignement Supérieur et de la Recherche

Centre Universitaire Abdelhafid Boussouf - Mila
Institut des Mathématiques et Informatique
Département d'informatique

Mémoire préparé pour obtenir le diplôme de Master en Informatique

Option : Sciences et Technologies de l'Information et de la Communication (STIC)

Thème :

Approche par apprentissage profond pour la sélection d'algorithmes dans l'optimisation à boîte noire (BBOP)

Soutenue devant le jury :

Réalisé par :
Wail Boudjema
Sedari Osama

Président : Mme Talai Meriem
Encadreur : Mme Afri faiza
Examinateur : Mme. Deffas Zineb

Année Universitaire : 2024/2025


بسم الله الرحمن الرحيم

Remerciements

On dit souvent que le voyage est aussi important que la destination.
Les cinq années que nous avons passées à l'université nous ont permis de comprendre pleinement la profondeur de cette citation.
Ce parcours n'a pas été exempt de défis ni de remises en question, qui ont nécessité de longues heures de travail afin de trouver les réponses appropriées.

En terminant ce travail, nous tenons tout d'abord à remercier Allah Tout-Puissant, qui nous a donné la foi et la force d'atteindre cette étape importante de notre parcours académique.

Nous exprimons également notre profonde reconnaissance à nos parents, pour leur soutien moral inconditionnel tout au long de nos études, ainsi que pour les innombrables sacrifices qu'ils ont consentis afin de nous offrir les meilleures conditions de réussite.
Leur encouragement constant a été pour nous une source de motivation inestimable.

Nous adressons aussi nos plus sincères remerciements à notre encadrant Mme Afri Faiza, pour avoir encadré et guidé nos travaux de recherche.
Ses conseils avisés, son soutien continu et ses remarques constructives ont été essentiels à l'amélioration de la qualité de notre mémoire.
Grâce à elle, nous avons beaucoup appris, tant sur le plan académique que personnel. Nous n'oublierons jamais son aide précieuse lors de la relecture et de la correction de notre travail.

Nos remerciements vont également à nos familles pour leur soutien moral indéfectible, ainsi qu'à nos amis, pour leur aide directe ou indirecte tout au long de ce parcours. Un grand merci à chacun de vous.

Enfin, nous souhaitons adresser notre profonde gratitude à l'ensemble des enseignants du Département d'Informatique de l'Université Abdelhafid Boussouf – Mila, pour la qualité de leur enseignement, leur accompagnement et leur dévouement.

Dédicace

À ma mère, dont le dévouement inébranlable et le soutien inconditionnel ont constitué les fondements mêmes de ma réussite.
Par son amour infini, ses sacrifices silencieux et ses conseils empreints de sagesse, elle a guidé chacun de mes pas sur le chemin de la connaissance et m'a toujours encouragé à poursuivre mes aspirations les plus profondes.
À travers ce travail, je tiens à lui exprimer ma reconnaissance éternelle pour sa présence constante dans ma vie, et pour être cette source inestimable de force, de réconfort et d'inspiration.

À mon père, dont la persévérance, l'intégrité et les sacrifices quotidiens ont été pour moi un exemple vivant de courage et de détermination.
Son travail acharné, accompli dans la discrétion et l'humilité, a tracé la voie vers mon succès.
Puisse Dieu le préserver, et que ce travail soit le reflet de sa foi en moi, de son soutien indéfectible et des valeurs nobles qu'il m'a transmises.

Osama Sedari

Dédicace

À ma mère, dont le dévouement inébranlable et le soutien inconditionnel ont constitué les fondements mêmes de ma réussite.
Par son amour infini, ses sacrifices silencieux et ses conseils empreints de sagesse, elle a guidé chacun de mes pas sur le chemin de la connaissance et m'a toujours encouragé à poursuivre mes aspirations les plus profondes.
À travers ce travail, je tiens à lui exprimer ma reconnaissance éternelle pour sa présence constante dans ma vie, et pour être cette source inestimable de force, de réconfort et d'inspiration.

À mon père, dont la persévérance, l'intégrité et les sacrifices quotidiens ont été pour moi un exemple vivant de courage et de détermination.
Son travail acharné, accompli dans la discrétion et l'humilité, a tracé la voie vers mon succès.
Puisse Dieu le préserver, et que ce travail soit le reflet de sa foi en moi, de son soutien indéfectible et des valeurs nobles qu'il m'a transmises.

Wail Boujemaa

Abstract

Selecting the most suitable algorithm to solve a continuous black-box optimization problem is a challenging task. With a wide range of optimization algorithms evaluated annually and easier access to black-box optimization functions—primarily due to the efforts of the COCO platform—an automated method for algorithm selection in single-objective black-box optimization problems has become necessary.
In this thesis, we leverage Deep Learning technology combined with Exploratory Landscape Analysis techniques to predict the optimal solver (SBS) for any given set of problems. The performance of the proposed Deep Learning-based model surpasses that of a model based on traditional machine learning algorithms when applied to a set of continuous black-box optimization problems, thereby demonstrating its effectiveness.

Keywords: Algorithm selection, Black-box optimization, Exploratory Landscape Analysis, Deep Learning, Single-objective continuous optimization, COCO platform.

Résumé

Sélectionner l'algorithme le plus approprié pour résoudre un problème d'optimisation continu à boîte noire est une tâche complexe. Avec une gamme étendue d'algorithmes d'optimisation évalués annuellement et un accès facilité aux fonctions d'optimisation à boîte noire, principalement grâce aux efforts de la plateforme COCO, une méthode automatisée de sélection d'algorithmes pour les problèmes d'optimisation à boîte noire à objectif unique est devenue nécessaire. Dans ce mémoire, nous utilisons la technologie d'apprentissage profond combinée à des techniques d'Analyse exploratoire du paysage pour prédire le solveur optimal (SBS) pour n'importe quel ensemble de problèmes donné. Les performances du modèle proposé basé sur l'apprentissage profond surpassent celles d'un modèle basé sur des algorithmes d'apprentissage automatique lorsqu'il est appliqué à un ensemble de problèmes d'optimisation à boîte noire continu, démontrant ainsi son efficacité.

Mots-clés **:** Sélection d'algorithmes, Optimisation à boîte noire, Analyse exploratoire du paysage, Apprentissage profond, Optimisation continue à objectif unique, plateforme COCO.

ملخص

تعد مهمه اختيار الخوارزميه الانسب لحل مشكله تحسين مستمره ذات صندوق اسود من المهام المعقده مع توفر مجموعه واسعه من خوارزميات التحسين التي يتم تقييمها سنويا وسهولة الوصول الى دوال التحسين ذات الصندوق الاسود وذلك بفضل جهود منصه COCO اصبحت هناك حاجه ملحه الى طريقه اليه لاختيار الخوارزميه المثلى لمشاكل التحسين دات الهدف الوحيد

في هذا العمل نعتمد على تقنيات التعلم العميق (Deep Learning) المدمجه مع تحليل المشهد الاستكشافي (ELA) بهدف التنبؤ بالمحلل الامثل (SBS) لاي مجموعه من المشاكل وقد اثبت النموذج المقترح المبني على التعلم العميق تفوقا ملحوظا على النماذج المبنية على خوارزميات التعلم الالي التقليديه عند تطبيقه على مجموعه من مشاكل التحسين المستمر ذات الصندوق الاسود مما يؤكد فعاليته

الكلمات المفتاحية **:** اختيار الخوارزميات، التحسين بصندوق أسود، تحليل المشهد الاستكشافي، التعلم العميق، التحسين المستمر بهدف وحيد، منصة COCO

Table des matières

Chapitre 2 : Sélection d'Algorithmes d'optimisation

Chapitre 3 : Construction de la base de données et mise en œuvre de la méthodologie de sélection d'algorithmes

Table des figures

Liste des tableaux

Liste des Acronymes

ELA	Exploratory Landscape Analysis
DL	Deep Learning
COCO	COmparing Continuous Optimizers
BBO	Black-Box Optimization
BBOP	Black-Box Optimization Problems
TPU	Tensor Processing Unit
GPU	Graphics Processing Unit
ASP	Algorithm Selection Problem
PCA	Principal Component Analysis
SMOTE	Synthetic Minority Over-sampling Technique
FLACCO	Feature-based Landscape Analysis of Continuous and Constrained Optimization
LHS	Latin Hypercube Sampling
ERT	Expected Running Time
RELERT	Relative Expected Running Time
SBS	Single Best Solver
CNN	Convolutional Neural Network
MLP	Multi-Layer Perceptron
SVM	Support Vector Machine

Introduction Générale

L’optimisation joue un rôle central dans presque tous les domaines de la science appliquée, de l’ingénierie à l’intelligence artificielle, en passant par la recherche opérationnelle, la finance ou encore la médecine. On y cherche toujours la meilleure solution possible à un problème donné, selon des critères de performance, de coût, de qualité, etc.

Mais dans de nombreux cas réels, la fonction à optimiser est inconnue, ou bien elle est si complexe qu’on ne peut ni en écrire une formule explicite ni en calculer les dérivées. On parle alors d’optimisation à boîte noire (black-box optimization**)**. L'exemple le plus courant est le réglage des hyperparamètres d’un réseau de neurones : on ne sait pas prédire exactement à l’avance la performance du modèle, on peut seulement la mesurer à travers une série d’expériences. De tels problèmes sont difficiles par nature : on doit se contenter de tester, d’observer, et de comparer les résultats sans connaître l'intérieur du système.

Face à cette incertitude, une question cruciale se pose : quel algorithme utiliser pour optimiser efficacement une fonction dont on ignore tout ?

Il existe une multitude d’algorithmes — évolutionnaires, swarm-based, bayésiens, etc. — chacun ayant ses forces et ses faiblesses. Un même algorithme peut exceller sur une catégorie de problèmes tout en échouant sur d’autres. Dès lors, le choix du bon algorithme devient un défi en soi, et c’est là qu’intervient le concept de sélection automatique d’algorithmes.

Notre travail s’inscrit dans cette problématique : comment automatiser intelligemment le choix d’un algorithme d’optimisation à partir d’un problème inconnu et sans connaissance préalable sur sa structure ?

Nous proposons une approche basée sur l’apprentissage profond (Deep Learning), combinée avec des techniques dites d’analyse exploratoire du paysage (Exploratory Landscape Analysis (ELA)), qui permettent de caractériser le problème d’optimisation par des métriques numériques issues d’un simple échantillonnage. Ces métriques vont jouer le

rôle de « carte d'identité » du problème, à partir de laquelle notre modèle pourra apprendre à reconnaître quel algorithme est le plus prometteur.

L'originalité de notre approche réside notamment dans l'utilisation d'un portefeuille étendu d'algorithmes, comprenant plus de quinze méthodes, allant des classiques tels que CMA-ES ou DE, à des variantes plus modernes comme L-SHADE, BIPOP, IPOP-CMA, et d'autres. Nous modélisons le choix algorithmique comme un problème de classification supervisée, dans lequel chaque instance du problème est associée à son solveur optimal connu. Par ailleurs, notre travail exploite des jeux de données de benchmark publics, notamment ceux fournis par la plateforme COCO, afin de garantir la reproductibilité et la pertinence des résultats obtenus.

Les objectifs principaux sont de concevoir un pipeline automatique de recommandation, d'évaluer l'efficacité des modèles de Deep Learning, de comparer avec des approches classiques

Nos contributions incluent la construction d'une base de données riche, la mise en œuvre de plusieurs architectures neuronales, la gestion des classes déséquilibrées.
Ce mémoire est structuré en quatre chapitres :
Le premier chapitre présente les fondements théoriques, en définissant l'optimisation à boîte noire, en exposant les principaux algorithmes, ainsi qu'en introduisant l'Exploratory Landscape Analysis (ELA) et le Deep Learning.
Le deuxième chapitre aborde la problématique de la sélection d'algorithmes (ASP). Il présente ensuite un état de l'art des différentes approches existantes, notamment AutoML et le méta-apprentissage, avant de procéder à une analyse critique des travaux connexes dans ce domaine

Le troisième chapitre porte sur la préparation du dataset à partir des benchmarks COCO et l'extraction des caractéristiques ELA. Aussi, il évoque la modélisation du problème en classification avec un modèle de Deep Learning, incluant la stratégie d'entraînement et de validation. Dans le quatrième chapitre, nous présentons les résultats, et les expérimentations, avec une comparaison de notre approche avec les algorithmes classiques. Nous terminons par une conclusion générale qui synthétise les contributions, les limites et propose des perspectives d'amélioration.

CHAPITRE

1

Fondement théorique

1.1.Introduction

Dans ce chapitre, nous abordons les bases théoriques sur lesquelles repose notre travail. Nous présentons d'abord le concept d'optimisation à boîte noire, un cadre fréquent dans les problèmes complexes où l'on ne dispose pas d'expression analytique de la fonction objectif. Ensuite, nous introduisons l'analyse exploratoire du paysage (ELA), une technique permettant de mieux comprendre la structure des fonctions d'optimisation. Enfin, nous explorons les fondements de l'apprentissage profond, outil clé de notre méthodologie.

1.2.Optimisation à boîte noire

1.2.1. Définition et principes généraux

L'optimisation à boîte noire (Black-Box Optimization, BBO) désigne un cadre dans lequel la fonction objective à optimiser est inconnue, non accessible analytiquement ou trop coûteuse à évaluer. L'évaluateur ne dispose d'aucune information interne sur la structure de la fonction : il ne peut qu'observer la sortie associée à une entrée donnée, sans dérivées ni équation explicite [1].

Ce type de configuration est fréquent dans des domaines complexes où les modèles sont fondés sur des simulations, des processus physiques ou des systèmes d'apprentissage automatique. Ainsi, la fonction à optimiser est vue comme une "boîte noire" dont on contrôle les entrées, mais dont on ignore le fonctionnement interne

1.2.2. Applications courantes

La BBO est particulièrement utile dans plusieurs contextes réels, notamment :

Réglage des hyperparamètres : dans l'apprentissage automatique, l'ajustement de paramètres tels que le taux d'apprentissage, le nombre de couches ou la taille des mini-batchs

relève d'un problème à boîte noire, car la relation entre paramètres et performance n'est ni linéaire ni explicite [2].

Simulation industrielle : dans les domaines tels que la mécanique des fluides (CFD), l'aérodynamique ou la finance, la fonction d'évaluation dépend de simulations coûteuses. L'optimisation se fait alors sans expression analytique, en exploitant uniquement les résultats des sorties simulées [3].

1.2.3. Algorithmes d'optimisation à boîte noire

Face à l'opacité de ces fonctions, les méthodes d'optimisation classiques (comme le gradient ou les méthodes convexes) deviennent inapplicables. À la place, des algorithmes évolutionnaires et inspirés de la nature sont souvent privilégiés. Parmi les plus utilisés :

a) **CMA-ES (Covariance Matrix Adaptation Evolution Strategy)** : un algorithme évolutionnaire qui ajuste dynamiquement la matrice de covariance d'une distribution de recherche pour guider efficacement la convergence [4].

b) **DE (Differential Evolution)** : une méthode basée sur la combinaison vectorielle de solutions existantes pour générer de nouveaux candidats, adaptée aux espaces non convexes et multimodaux.

c) **PSO (Particle Swarm Optimization)** : inspiré du comportement social des essaims (oiseaux, poissons), cet algorithme fait évoluer une population de solutions en tenant compte à la fois de l'expérience personnelle et collective [5].

Ces algorithmes illustrés dans la figure 1 sont évalués sur des plateformes de benchmarking standardisées comme COCO [6] (Comparing Continuous Optimizers), qui fournissent des fonctions tests et des indicateurs objectifs pour mesurer la performance.

Figure 1: Optimisation boite noire

1.3. Benchmarking des algorithmes d'optimisation

1.3.1 Objectif du benchmarking

Le benchmarking est un processus d'évaluation comparative visant à analyser les performances de différents algorithmes d'optimisation sur un ensemble représentatif de problèmes [54]. Dans le contexte de la sélection d'algorithmes, il constitue un outil essentiel pour :

- Identifier les méthodes les plus performantes dans divers contextes
- Comprendre les conditions dans lesquelles chaque algorithme excelle ou échoue
- Développer des systèmes de recommandation algorithmiques adaptés à de nouvelles instances.

1.3.2 Étapes du processus de benchmarking

Le benchmarking rigoureux d'algorithmes suit généralement plusieurs étapes clés [54] :

(a) Constitution d'un ensemble de problèmes

Diversité : sélection d'instances variées, représentatives de différentes structures de fonctions (unimodales, multimodales, bruyantes, dégénérées, etc.).

Représentativité : couverture équilibrée des caractéristiques rencontrées en pratique (taille, difficulté, domaine) [*].

(b) Choix des algorithmes à tester

Pluralité d'approches : inclusion d'algorithmes issus de familles différentes (heuristiques, évolutionnaires, essaim, bayésiens…).

Multiples configurations : exploration de différentes combinaisons d'hyperparamètres pour chaque algorithme.

(c) Définition des métriques de performance

Qualité des solutions : écart à la valeur optimale, taux de réussite.

Efficacité : nombre d'évaluations, temps de calcul, consommation mémoire.

Robustesse : stabilité des performances sur différentes instances.

(d) Méthodes d'analyse et de validation

Validation croisée : séparation des jeux d'entraînement/test pour éviter le surajustement.

Analyse statistique : comparaison rigoureuse à l'aide de tests (Wilcoxon, Friedman, etc.).

(e) Production de rapports et interprétation

Visualisations : courbes de convergence, heatmaps de performance, distribution des résultats (cf. Figure 1.1).

Interprétabilité : identification des relations entre les caractéristiques des problèmes et les performances algorithmiques.

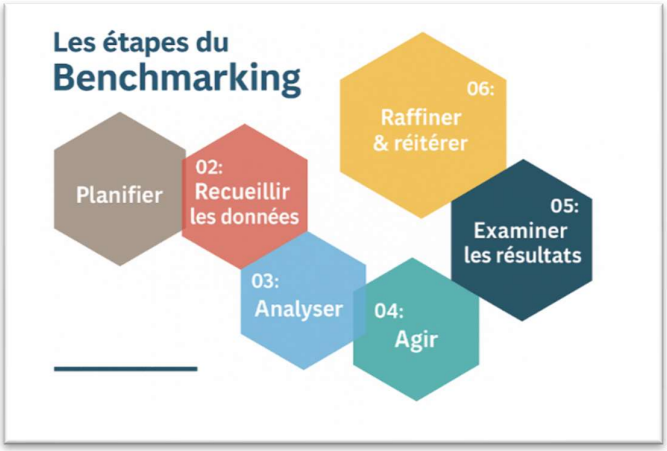


Figure 2: Schéma du processus de benchmarking

1.3.3 Cadres de benchmarking standardisés

Pour garantir la reproductibilité et la comparabilité des expériences, plusieurs plateformes ont été développées. Parmi les plus utilisées, on trouve :

ASlib (Algorithm Selection Library) : une bibliothèque pour la sélection d'algorithmes, contenant des jeux de données, des caractéristiques de problèmes et des résultats expérimentaux.

OpenML, **BBComp**, et surtout **COCO** qui est abordé plus en détail ci-dessous.

1.3.4 Le cadre COCO (Comparing Continuous Optimizers)

COCO est une plateforme open source dédiée au benchmarking rigoureux des algorithmes d'optimisation continue. Elle fournit un ensemble standardisé de **fonctions de test** couvrant divers degrés de difficulté (rugosité, séparabilité, modularité, etc.), des **indicateurs objectifs** comme le *runtime* (ERT : Expected Running Time), les taux de réussite, ou les distributions de performances, des **outils d'analyse et de visualisation** automatisés pour générer des rapports reproductibles.

Grâce à son infrastructure modulaire, COCO permet aux chercheurs de :

- Simuler des situations réalistes d'optimisation à boîte noire,
- Comparer leurs méthodes à des **références communautaires**,
- Exploiter un grand nombre d'exécutions pour obtenir des résultats statistiquement significatifs [2].

La figure 3 présente le flux de travail du cadre d'expérimentation COCO (COmparing Continuous Optimizers) pour l'évaluation comparative de solveurs d'optimisation en boîte noire. Les étapes principales sont :

- Intégration multi-langages : Les solveurs fournis par l'utilisateur peuvent être écrits en C/C++, Python, Java, Matlab/Octave.
- Expérimentation COCO : Les expériences sont réalisées à l'aide de groupes de tests (comme bbob, bbob-biobj, etc.), et incluent un système de journalisation permettant d'enregistrer les performances du solveur.

– Traitement post-expérimental (post-processing) : Les fichiers de journaux (log files) issus de ces expérimentations sont utilisés pour générer diverses formes de résultats : Graphiques (Plots), tables de résultats, pages HTML interactives pour la visualisation des résultats, modèles LaTeX pour la génération automatique de rapports.

L'ensemble du processus permet de standardiser la comparaison, l'analyse et la visualisation des performances de différents algorithmes d'optimisation, tout en facilitant l'intégration de nouveaux solveurs grâce à son interface multi-langages

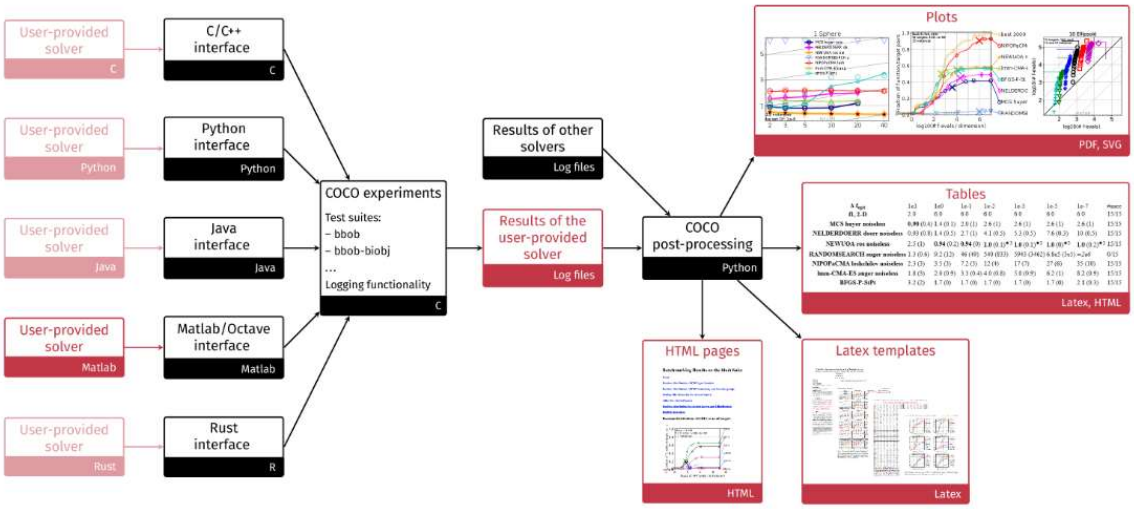


Figure 3: Structure générale du projet COCO

1.4. Analyse exploratoire du paysage (ELA)

1.4.1 Définition et objectif

Le terme analyse exploratoire du paysage (ELA) [7] désigne un ensemble de caractéristiques qui quantifient certaines propriétés d'un problème d'optimisation continue.

Dans sa version initiale, l'ELA comprenait un total de 50 caractéristiques, regroupées en six catégories dites de bas niveau : Convexité, Courbure, Distribution des valeurs de y, Niveaux

de contour (Levelset), Recherche locale et Modèle méta (Meta Model).

Ces valeurs numériques servaient à caractériser des propriétés de haut niveau, généralement catégorielles et définies par des experts, telles que la structure globale, la multimodalité ou la variation d'échelle des variables [8].

La figure 4 illustre les liens entre les propriétés de bas niveau et les propriétés de haut niveau.

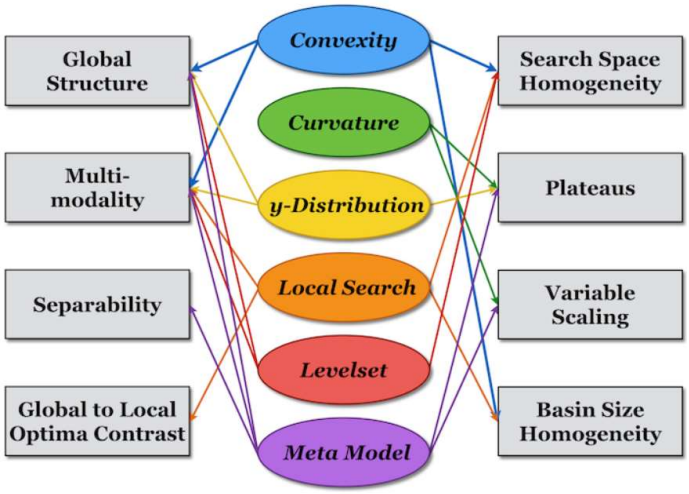


Figure 4: Une description détaillée des caractéristiques ELA

L'objectif principal de l'ELA est de fournir des indices sur les propriétés du problème (multimodalité, convexité, régularité…) afin d'orienter le choix des algorithmes d'optimisation les plus adaptés. En ce sens, elle constitue une brique essentielle dans les systèmes de *meta-apprentissage* ou de *sélection automatique d'algorithmes*.

1.4.2 Fonctionnement général de l'ELA

Le processus de l'ELA repose sur trois étapes principales :

Échantillonnage du paysage

Un ensemble de points est généré (souvent aléatoirement ou selon un plan de sampling stratifié) dans l'espace de recherche, et la fonction objectif est évaluée à chacun de ces points.

Extraction de descripteurs

À partir des données obtenues (coordonnées + valeurs de la fonction), un ensemble de caractéristiques statistiques et structurelles est calculé.

Utilisation des descripteurs

Ces descripteurs peuvent ensuite être utilisés pour **comparer** des fonctions entre elles. Aussi

pour **prédire** la performance relative d'un algorithme sur un nouveau problème, ou comme **entrée d'un modèle d'apprentissage** (ex. : réseau de neurones) pour la sélection automatique d'un algorithme.

1.4.3 Types de descripteurs

Les descripteurs extraits via l'ELA couvrent plusieurs dimensions :

- **Statistiques de base** : Moyenne, écart-type, asymétrie (*skewness*), aplatissement (*kurtosis*) des valeurs de la fonction.
- **Mesures de rugosité** (*ruggedness*) : Évaluent la présence de nombreux minima locaux et la difficulté à naviguer dans le paysage.

Propriétés topologiques et géométriques : Indiquent si le paysage est *convexe*, *multimodal*, *séparable* ou *composé de plateaux*.
Descripteurs basés sur la dérivabilité locale (quand possible) : Estimation de la pente moyenne, de la linéarité locale, etc.
Chaque descripteur apporte un éclairage complémentaire sur la structure du problème, et leur combinaison permet de constituer une **empreinte numérique** du paysage d'optimisation.

1.4.4 Utilisation dans la sélection d'algorithmes

L'un des usages les plus pertinents de l'ELA est sa combinaison avec des techniques d'apprentissage automatique pour prédire l'algorithme le plus performant sur une instance donnée. On parle alors de **meta-modélisation** ou de **meta-learning** guidé par des descripteurs de paysage.
L'ELA est ainsi devenue une composante centrale des systèmes intelligents d'optimisation, en fournissant une base de connaissance **interprétable**, **quantifiable** et **généralisable**.

1.5. Apprentissage profond

1.5.1 Définition générale

L'apprentissage profond (*Deep Learning*) est une sous-discipline de l'apprentissage automatique (*machine learning*) qui repose sur l'utilisation de **réseaux de neurones artificiels à plusieurs couches**. Ces architectures permettent de modéliser des relations complexes, non linéaires et hiérarchiques entre des données d'entrée et une sortie cible [9].

Contrairement aux modèles classiques, l'apprentissage profond excelle dans l'extraction automatique de représentations pertinentes à partir de données brutes, sans nécessiter d'ingénierie manuelle avancée des caractéristiques.

Dans le contexte de ce travail, l'apprentissage profond est utilisé comme un outil prédictif pour **apprendre à associer les caractéristiques ELA d'un problème d'optimisation à l'algorithme le plus performant**.

1.5.2 Structure d'un réseau de neurones

Un réseau de neurones profond (*deep neural network*) est constitué de plusieurs couches interconnectées, chacune remplissant une fonction spécifique [9] :

Couche d'entrée :

Reçoit les données initiales. Dans notre cas, il s'agit des descripteurs ELA extraits d'un problème d'optimisation.

Couches cachées (*hidden layers*) :

Effectuent des transformations successives via des fonctions d'activation non linéaires (ReLU, sigmoid, etc.). Chaque couche apprend une représentation de plus en plus abstraite des données.

Couche de sortie :

Produit la sortie finale du modèle. Dans notre étude, cela peut correspondre à une c**lassification** (choix d'un algorithme) ou à une **régression** (performance attendue d'un algorithme donné).

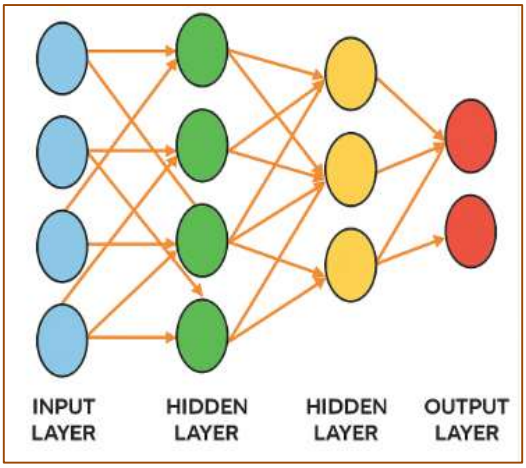


Figure 5: Architecture simplifiée d'un réseau de neurones artificiels

1.5.3 Processus d'entraînement

L'entraînement d'un réseau de neurones consiste à ajuster les poids des connexions entre les neurones afin de minimiser une **fonction de perte**, qui mesure l'écart entre la sortie produite et la sortie attendue.

Le processus comprend généralement les étapes suivantes :

- **Propagation avant** (*forward pass*) : les données d'entrée sont transmises couche par couche pour générer une prédiction.
- **Calcul de l'erreur** : la fonction de perte (ex. : erreur quadratique, entropie croisée) mesure la qualité de la prédiction.
- **Rétropropagation** (*backpropagation*) : l'erreur est propagée en sens inverse dans le réseau pour calculer les gradients.
- **Mise à jour des poids** : les poids sont ajustés à l'aide d'un algorithme d'optimisation tel que la descente de gradient, Adam ou RMSprop.

Ce processus est répété sur plusieurs itérations (ou *époques*) jusqu'à convergence vers un minimum de la fonction de perte [10].

1.5.4 Facteurs influençant la qualité de l'apprentissage

Deux éléments majeurs conditionnent la performance d'un réseau de neurones :

- **La qualité et la quantité des données** :
  Un grand nombre de données variées et représentatives permet d'améliorer la capacité de généralisation du modèle et de réduire le risque de surapprentissage (*overfitting*). À l'inverse, des données bruitées ou trop rares peuvent nuire à l'apprentissage [11].
- **La puissance de calcul disponible** :
  L'entraînement de réseaux profonds nécessite des ressources importantes, notamment pour traiter de grands volumes de données ou pour optimiser des architectures complexes. L'usage de processeurs graphiques (GPU) ou de TPU est courant pour accélérer le calcul.

L'apprentissage profond constitue une solution particulièrement efficace pour capturer des relations complexes entre les caractéristiques d'un problème d'optimisation (via l'ELA) et le comportement des algorithmes. Cette approche ouvre la voie à la mise en place de systèmes intelligents de **sélection automatique d'algorithmes [11]**.

1.6 Conclusion

Dans ce premier chapitre, nous avons établi les fondements théoriques sur lesquels repose notre démarche méthodologique. Nous avons tout d'abord introduit le concept d'**optimisation à boîte noire**, ensuite nous avons examiné le rôle du **benchmarking** dans l'évaluation systématique de ces algorithmes, en détaillant ses étapes clés, ses indicateurs de performance et les plateformes de référence telles que **COCO**, qui offrent un cadre rigoureux pour la comparaison expérimentale.

L'**analyse exploratoire du paysage (ELA)** a ensuite été présentée comme une approche innovante permettant de caractériser automatiquement la structure des fonctions à optimiser. Enfin, nous avons introduit les **principes de l'apprentissage profond**, en mettant en lumière son rôle central dans notre approche.

Le chapitre suivant sera dédié à la présentation du cadre expérimental, des jeux de données utilisés, ainsi que des paramètres retenus pour l'entraînement et l'évaluation des modèles.

CHAPITRE

2

Sélection d'Algorithmes d'optimisation

2.1.Introduction

Dans le chapitre précédent, nous avons présenté les fondements de l'optimisation à boîte noire ainsi que la diversité des algorithmes pouvant être mobilisés. Toutefois, une question centrale reste posée : **comment choisir automatiquement le meilleur algorithme pour un problème donné**, surtout lorsqu'on ne dispose d'aucune connaissance a priori sur sa structure ?

Cette question est au cœur du **problème de sélection d'algorithmes** (*Algorithm Selection Problem*, ou **ASP**), une sous-discipline du *métapprentissage* (*meta-learning*) qui vise à automatiser ce choix de manière fiable et efficace [12].

2.2 Définition du problème de sélection d'algorithmes (ASP)

Le Problème d'Algorithme Sélection (ASP) a été formellement défini par John Rice en 1976. Il s'agit d'un processus décisionnel visant à choisir, pour un problème donné, l'algorithme le plus performant parmi un ensemble prédéfini [13].

D'un point de vue formel, il s'agit d'apprendre une fonction :

où chaque problème $p \in P$ est associé à un algorithme $a \in A$ qui minimise une fonction de coût (temps d'exécution, erreur, consommation de ressources, etc.) [13].

La sélection dépend principalement de deux éléments :

- Les caractéristiques du problème (appelées *features* ou *descripteurs*),
- Les performances observées ou prédites des algorithmes sur des problèmes similaires.

2.3 Stratégies de sélection : par ensemble vs. par instance

Deux approches principales peuvent être distinguées dans la résolution du ASP :

2.3.1 Sélection par ensemble (per-set selection)

Il s'agit de sélectionner **un algorithme unique** qui sera utilisé pour toutes les instances d'un ensemble donné. Cette stratégie est simple à implémenter et courante dans les benchmarks (ex. : COCO), mais elle ne tient pas compte des variations entre les instances.

***Exemple** : utiliser DE ou PSO comme algorithme par défaut, indépendamment de la nature du problème.*

2.3.2 Sélection par instance (per-instance selection)

Cette approche plus fine consiste à **choisir dynamiquement un algorithme adapté à chaque instance**, en s'appuyant sur ses caractéristiques. Elle permet d'améliorer significativement les performances globales, au prix d'une analyse préalable plus poussée [15].

2.4 Méthodes de résolution du problème de sélection d'algorithmes

Différentes méthodes ont été développées pour résoudre l'ASP. Toutes reposent sur le principe d'**extraire des descripteurs de l'instance**, puis **prédire la performance relative de chaque algorithme**.

2.4.1 AutoML (Automated Machine Learning)

L'AutoML vise à automatiser l'ensemble du processus d'apprentissage automatique, incluant la sélection d'algorithmes. Il s'appuie sur :

- Des portefeuilles d'algorithmes préconfigurés (ex. : Auto-sklearn),
- Des techniques de recherche (ex. : optimisation bayésienne),
- Des heuristiques de sélection intelligentes.

Bien que très répandu, AutoML reste souvent générique **et n'est pas toujours adapté aux** problèmes d'optimisation continue à boîte noire [16].

2.4.2 Modèles de substitution (Surrogate models)

Les modèles de substitution consistent à prédire la performance d'un algorithme sans l'exécuter réellement, en s'appuyant sur un modèle prédictif (régression, réseau de neurones, forêt aléatoire…). Cela permet de réduire considérablement les coûts d'expérimentation [17]. La limite de ses modèles est que la qualité du modèle dépend fortement de la pertinence des descripteurs utilisés.

2.4.3 Métapprentissage (Meta-learning)

Le métapprentissage repose sur l'idée que les performances passées des algorithmes peuvent guider les choix futurs**.** Le processus suit généralement trois étapes :

1. Observation des performances sur un ensemble de problèmes,
2. Apprentissage d'une fonction de prédiction : caractéristiques ↔ performance,
3. Application de cette fonction à de nouveaux problèmes [18].

C'est cette approche qui constitue le noyau méthodologique de notre mémoire.

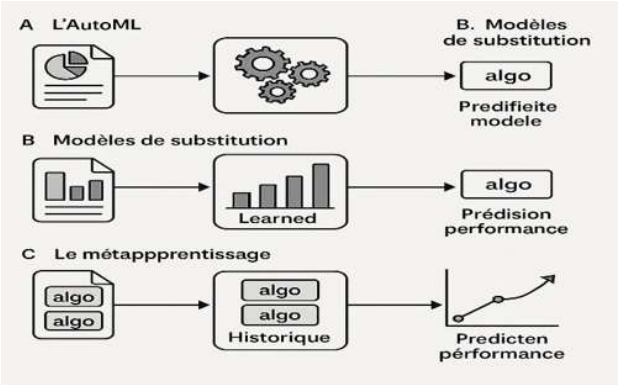


Figure 6 : Principales approches de la sélection d'algorithmes

2.5 Revue des travaux existants

Voici quelques contributions représentatives dans le domaine, illustrant l'évolution des approches basées sur les descripteurs ELA et le deep learning présenté dans le tableau 1 :

Contributions clés	Domaine de BBOP vise	Méthode Deep Learning utilisée	Année	Auteur(s) et Référence
Prédiction de la performance des algorithmes via features ELA + DL [19]	COCO / BBOB (Optimisation continue à boîte noire)	Réseaux de neurones entièrement connectés (MLP)	2020	**Krejca et al.** (NeurIPS Workshop
Sélection dynamique de solveurs pour chaque instance en temps réel [20]	BBOP dans les systèmes multi-agents	CNN + LSTM	2021	**Sun et al.** (Expert Systems)
Recommandation d'algorithmes basée sur une compression non linéaire des meta-features [21]	Hyperparameter optimization (boîte noire)	Autoencoders + Regressions profondes	2019	**Wistuba et al.** (MLJ)
Choix de l'algorithme optimal par acquisition basée DL [22]	Fonction objectif boîte noire (non dérivable)	Deep Bayesian Optimization	2017	**Belkhir et al.** (GECCO)

Tableau 1: Quelques travaux réalisés dans le domaine

2.6 Conclusion

Ce chapitre a permis d'introduire le **problème de sélection d'algorithmes**, en exposant ses fondements théoriques, ses stratégies de mise en œuvre, ainsi que les approches les plus prometteuses issues de la recherche actuelle.

Nous avons souligné l'intérêt croissant pour la **sélection par instance**, qui tire parti de l'analyse des caractéristiques des problèmes et du métapprentissage. C'est dans ce cadre que s'inscrit notre méthodologie, qui propose d'exploiter les descripteurs ELA combinés à un **modèle profond (DNN)** pour recommander l'algorithme optimal.

Avant de présenter cette méthodologie, nous devons d'abord détailler le **jeu de données**, les **fonctions test** et les **mesures de performance** utilisées dans l'étude. Cela fera l'objet du **chapitre 3**.

CHAPITRE 3

Construction de la base de données et mise en œuvre de la méthodologie de sélection d'algorithmes

3.1. Introduction

Dans les systèmes intelligents de sélection d'algorithmes, la qualité de la base de données utilisée pour entraîner les modèles prédictifs joue un rôle central. Une base de données bien construite permet de mieux capturer la diversité des problèmes d'optimisation et d'améliorer la précision du modèle lors du choix du meilleur algorithme à appliquer à un problème donné.

Dans ce chapitre, nous détaillons les étapes suivies pour construire notre base de données depuis la collecte des données brutes de performance des algorithmes, jusqu'à l'obtention d'un fichier final prêt à être utilisé. Ensuite, nous présentons le cadre méthodologique adopté pour sélectionner l'algorithme d'optimisation le plus adapté à l'aide de techniques d'apprentissage automatique.

3.2 Construction de la base de données

3.2.1 Benchmarks BBOB/COCO

Les données utilisées dans ce projet proviennent de la plateforme COCO (Comparing Continuous Optimizers)**.** COCO est un cadre de benchmark qui fournit des suites de fonctions de test standards, appelées BBOB (Black-Box Optimization Benchmarking), pour évaluer la performance des algorithmes d'optimisation continue. Les fonctions sont disponibles dans différentes dimensions $D \in \{2,3,5,10\}$ et pour plusieurs instances $IID \in \{1,...,15\}$ afin de capturer la variabilité stochastique.

Chaque fonction f appartient à un ensemble de 24 fonctions de test ***FID ∈{1,...,24}***, chacune représentant un paysage d'optimisation aux propriétés particulières (multimodalité, séparabilité, conditionnement, etc.) [24].

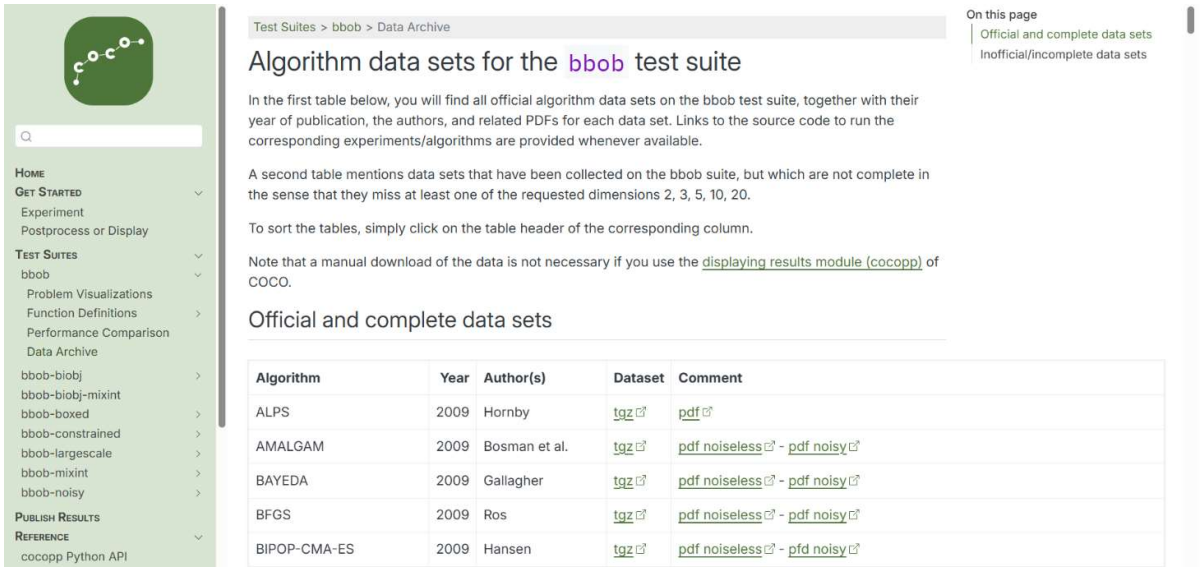


Figure 7: La base de données BBOB fournis par l'archive de COCO [24].

Chaque fichier contient :

- Le nombre d'évaluations FEi
- La valeur de fitness obtenue f***(x)***
- La réussite ou non d'atteindre

3.2.2 Portefeuille d'algorithmes sélectionnés

Au début de notre étude, nous avons collecté l'ensemble des algorithmes disponibles sur la plateforme GitHub BBOB, en filtrant uniquement ceux appartenant aux familles CMA-ES et DE ainsi que leurs variantes (plus de 120 algorithmes). Ce filtrage s'est basé sur la présence de mots-clés dans les noms des algorithmes, tels que *CMA*, *CMA-ES*, *DE*, *DE-rand*, etc.
Ce premier filtrage a permis de réduire le jeu de données à environ 50 algorithmes, dont les performances étaient jugées prometteuses.

Toutefois, en raison de la complexité croissante du jeu de données et de la nécessité de garantir une analyse efficace, nous avons ensuite sélectionné un sous-ensemble plus restreint, composé de 10 algorithmes représentatifs et performants. Ces algorithmes constituent la base

expérimentale de notre travail. Le tableau ci-dessous, offre une brève description pour chaque algorithme :

Nom de l'algorithme	**Description**
BIPOP-CMA-ES	Une variante du CMA-ES utilisant une stratégie bi-populationnelle (BIPOP), combinant deux régimes de taille de population pour une exploration plus efficace. Adaptée aux fonctions non convexes et multimodales
CMA-CSA-Atamna	Extension du CMA-ES incorporant la Cumulative Step-size Adaptation (CSA), qui ajuste dynamiquement le pas de mutation pour améliorer la convergence, notamment sur des espaces complexes ou mal conditionnés.
CMAES-APOP-KMA_Nguyen	Version spécialisée combinant : • APOP : adaptation dynamique de la taille de population selon la progression, • KMA : une technique basée sur la divergence de Kullback-Leibler pour guider la recherche. Optimisée pour les espaces stochastiques et multimodaux.
DE-BFGS-Voglis-Noiseless	Hybride combinant l'algorithme évolutionnaire DE pour l'exploration globale et la méthode quasi-Newton BFGS pour la recherche locale précise, conçu pour des fonctions sans bruit (déterministes).
a-CMA-ES	Variante adaptative du CMA-ES avec mécanismes améliorant la flexibilité du modèle de covariance pour une meilleure adaptation aux paysages complexes.
ad-CMA-ES_Gissler	Version du CMA-ES développée par Gissler, introduisant des adaptations dynamiques supplémentaires pour la matrice de covariance afin d'améliorer la convergence.
adm-CMA-ES_Gissler	Extension de ad-CMA-ES intégrant des mécanismes de contrôle adaptatifs renforcés pour la gestion du pas de mutation et de la matrice de covariance.

dm-CMA-ES_Gissler	Variante modifiée visant à améliorer la robustesse dans les environnements bruités ou fortement non convexes, avec ajustements spécifiques des paramètres dynamiques.
s-CMA-ES_Gissler	Version simplifiée de CMA-ES par Gissler, allégeant certains calculs tout en conservant une bonne capacité d'adaptation.
sd-CMA-ES_Gissler	Variante « simple et dynamique » intégrant des stratégies pour accélérer la convergence tout en maintenant la stabilité dans des paysages d'optimisation complexes.

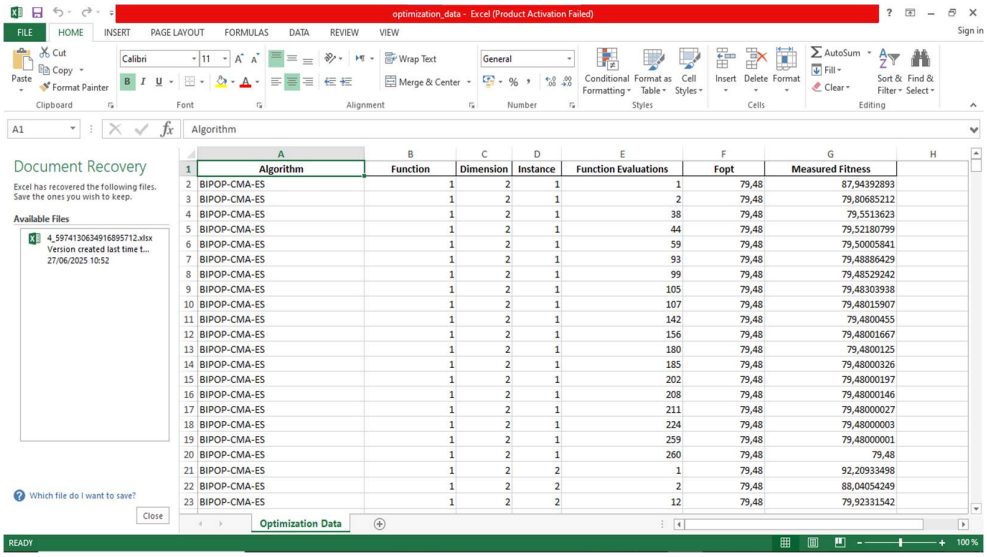
Tableau 2: Description des algorithmes du portefeuille sélectionné

3.2.3 Métriques de performance

Pour l'évaluation des performances, nous avons utilisé des métriques spécifiques, notamment l'Expected Running Time (ERT)

a) ERT (Expected Runtime to Target)

L'*ERT* est défini comme le nombre moyen d'évaluations nécessaires pour atteindre une précision donnée ε =10−2, sur un ensemble d'instances.



Algorithm	Function	Dimension	Instance	Function Evaluations	Fopt	Measured Fitness
BIPOP-CMA-ES	1	2	1	1	79,48	87,94392893
BIPOP-CMA-ES	1	2	1	2	79,48	79,80685212
BIPOP-CMA-ES	1	2	1	38	79,48	79,5513623
BIPOP-CMA-ES	1	2	1	44	79,48	79,52180799
BIPOP-CMA-ES	1	2	1	59	79,48	79,50005841
BIPOP-CMA-ES	1	2	1	93	79,48	79,48886429
BIPOP-CMA-ES	1	2	1	99	79,48	79,48529242
BIPOP-CMA-ES	1	2	1	105	79,48	79,48303938
BIPOP-CMA-ES	1	2	1	107	79,48	79,48015907
BIPOP-CMA-ES	1	2	1	142	79,48	79,4800455
BIPOP-CMA-ES	1	2	1	156	79,48	79,48001667
BIPOP-CMA-ES	1	2	1	180	79,48	79,4800125
BIPOP-CMA-ES	1	2	1	185	79,48	79,48000326
BIPOP-CMA-ES	1	2	1	202	79,48	79,48000197
BIPOP-CMA-ES	1	2	1	208	79,48	79,48000146
BIPOP-CMA-ES	1	2	1	211	79,48	79,48000027
BIPOP-CMA-ES	1	2	1	224	79,48	79,48000003
BIPOP-CMA-ES	1	2	1	259	79,48	79,48000001
BIPOP-CMA-ES	1	2	1	260	79,48	79,48
BIPOP-CMA-ES	1	2	2	1	79,48	92,20933498
BIPOP-CMA-ES	1	2	2	2	79,48	88,04054249
BIPOP-CMA-ES	1	2	2	12	79,48	79,92331542

Figure 8 ERT exemple de calcul

$$ERT(\varepsilon) = \frac{\sum_{i=1}^{n} FE_i(\varepsilon)}{\sum_{i=1}^{n} Succès_i(\varepsilon)}$$

où ***FEi(ε)*** est le nombre d'évaluations jusqu'à atteindre f(x) ≤ fopt+ ε **Succes*i*** ∈ {0,1}

Succès indique si l'objectif est atteint.

b) RELERT (Relative ERT) :

Le **RELERT** compare l'efficacité d'un algorithme donné à celle du meilleur sur une même instance.

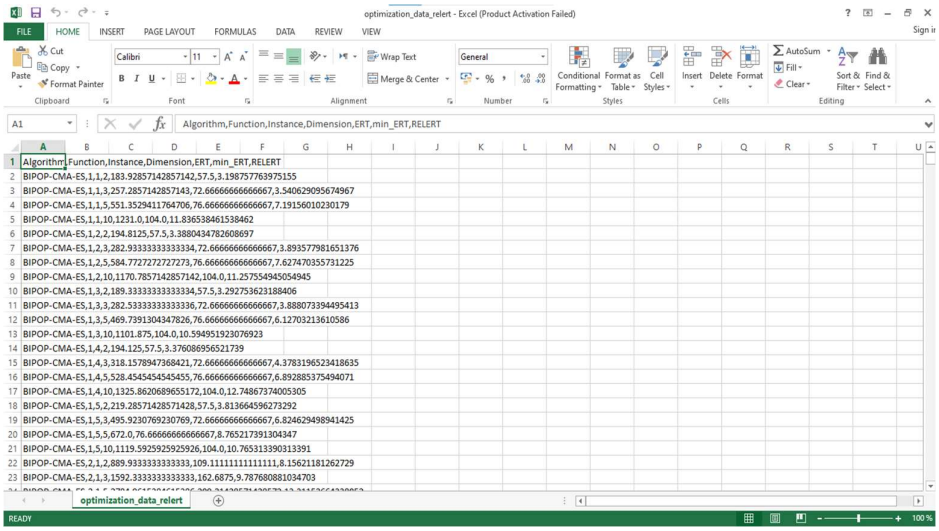


Figure 9: Résultats ERT relatif pour Divers Algorithmes

$$RELERT_{A,F,D} = \frac{ERT_{A,F,D}}{\min_{A'}(ERT_{A',F,D})}$$

L'algorithme ayant le RELERT le plus bas pour une instance donnée est désigné comme le **SBS (Single Best Solver)**.

3.3 Extraction des caractéristiques

3.3.1 Descripteurs ELA (via Flacco)

Nous avons utilisé l'outil *Flacco GUI* pour extraire automatiquement *151 caractéristiques* réparties en 6 familles :

- Caractéristiques de convexité
- Courbure locale
- Distribution de *f(x)*

- Recherche locale
- Méta-modélisation
- Analyse des ensembles de niveau

Méthode d'échantillonnage :

- Domaine : [−5,5] D
- Méthode : Latin Hypercube Sampling (LHS)
- Taille de l'échantillon : 50×D

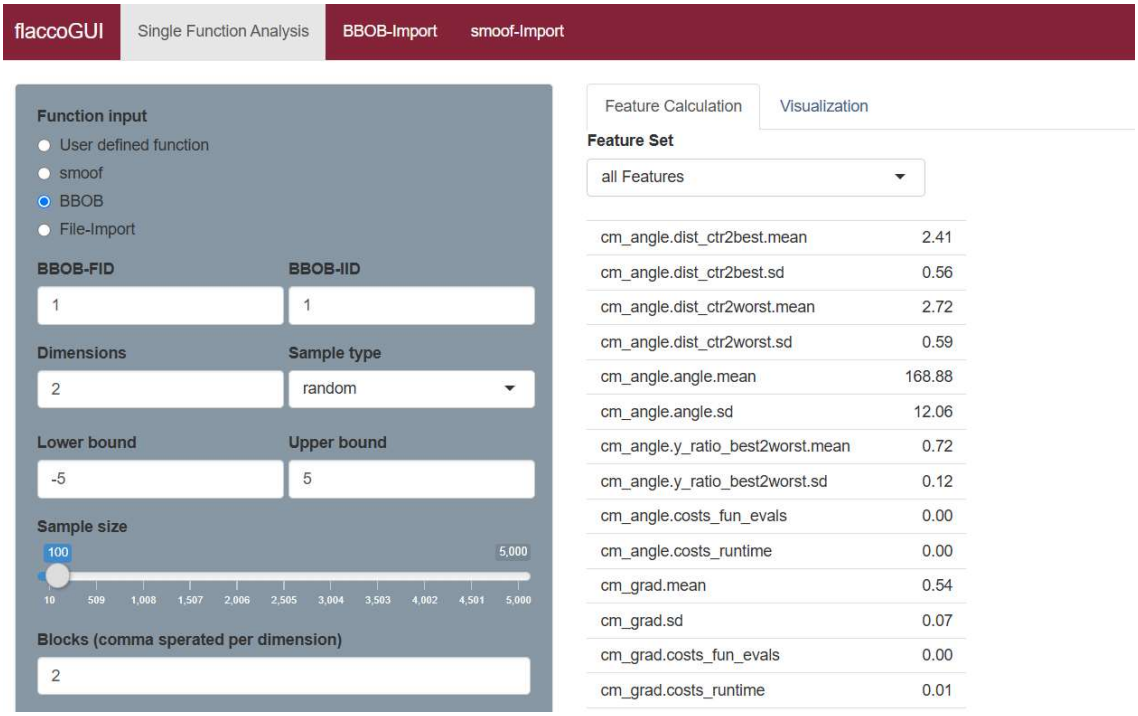


Figure 10: l'interface graphique Flacco [27].

Les caractéristiques ELA permettent de décrire le paysage des fonctions de manière quantitative (statistiques, géométrie, informations de dispersion, etc.). Ces caractéristiques sont extraites automatiquement via des bibliothèques telles que *Flacco Gui*. Elles constituent le vecteur d'entrée principal des modèles de prédiction [26].

3.3.2 Réduction de dimension

Des techniques comme PCA ou le filtrage par variance ont été appliquées pour réduire la redondance et améliorer la généralisation [27].

3.4 Prétraitement des données

3.4.1 Fusion et alignement

Les fichiers de performance sont associés aux fichiers de caractéristiques via les identifiants Function (FID)**,** Instance (IID)**,** et Dimension (D). Cette jointure garantit que chaque ligne du jeu de données fusionné représente une situation-problème unique [28].

3.4.2 Nettoyage et traitement

Les étapes suivantes ont été appliquées pour garantir la cohérence du dataset:

- Remplacement des valeurs infinies par un seuil maximal ou NaN,
- Imputation par la médiane pour les valeurs manquantes,
- Standardisation (moyenne 0, écart-type 1),
- Normalisation [0, 1] avec MinMaxScaler,
- Remplacement final des NaN /infinis restants par 0.
- Cette étape est cruciale pour éviter les biais lors de l'entraînement du modèle [28].

Algorithme 1 : Prétraitement

```
Fonction PRÉTRAITER(Performance,
  Caractéristiques)
  données1 ← CHARGERDONNÉES(Perfor-
  données2 ← CHARGERDONNÉES(Carac-
  téristiques)
  données1 ← CALCULERRELERT(données1)
  données ← FUSIONNERDONNÉES(données1
  données ← FILTRERDONNÉES(données)
  données ← NETTOYERDONNÉES(données)
  X ← données.SUPPRIMER_COLONNES
    ["Cible"])
  Y ← données["Cible"]
  X ← NORMALISERDONNEES(X)
  Y ← ENCODERCIBLE(Y)
  X_échantillonné, Y_échantillonné
    – APPLIQUERSMOTE(X, Y)
  retourner X_échantillonné, Y_échantillonné
  Fin Fonction
```

Figure 11: Pseudo-code pour la phase de prétraitement

3.4.3 Équilibrage des classes (SMOTE)

Les classes correspondant à certains algorithmes étaient sous-représentées. Pour équilibrer la distribution, la technique *SMOTE (Synthetic Minority Over-sampling Technique)* a été

appliquée. Elle génère de nouvelles instances synthétiques basées sur les plus proches voisins des exemples minoritaires. Cette étape est cruciale pour éviter les biais lors de l'entraînement du modèle [28].

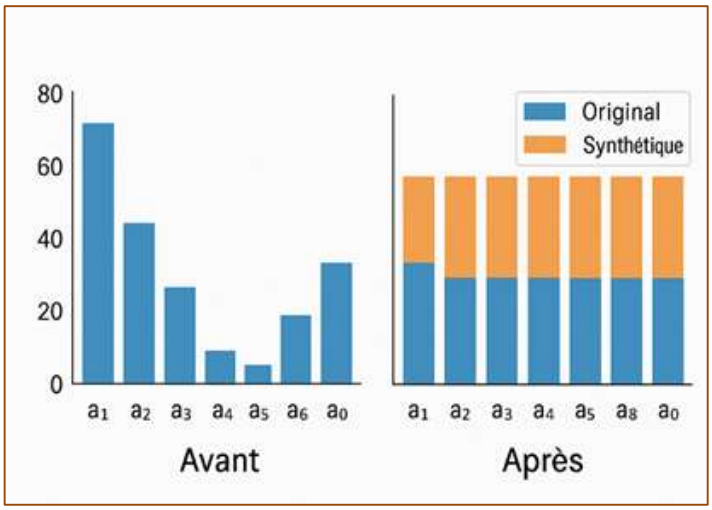


Figure 12: Equilibrage des classes avec SMOTE.

3.4.4 Composition finale

La Base de données finale contient :

- Des *features* normalisées issues de l'ELA,
- Une étiquette (label) correspondant à l'algorithme optimal, encodée via *LabelEncoder*,
- Un nombre réduit d'algorithmes (top 10) sélectionnés selon leur ERT moyen.

Base de Données	Nombre de lignes	Nbr de caractéristiques
Copie Finale	480	151

Tableau 3: Tableau montrant la composition de la base de données

Les objets de prétraitement (scalers, encodeurs) sont sérialisés pour assurer leur réutilisabilité dans les phases de validation et d'inférence.

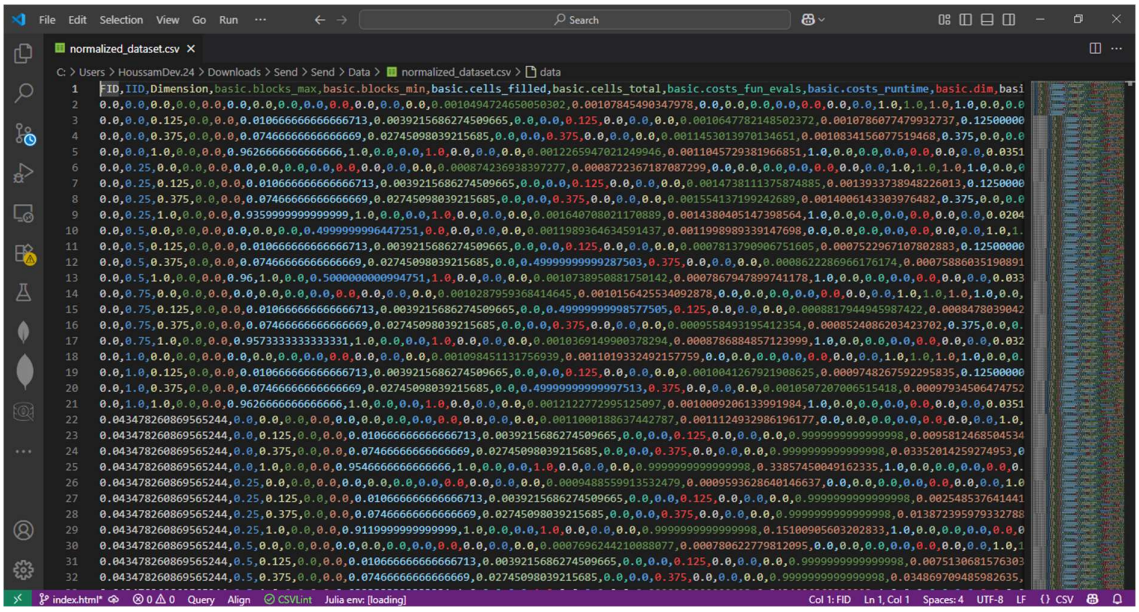


Figure 13: Base de données utilisée pour l'apprentissage.

3.5 Formulation du problème de sélection

3.5.1. Formulation comme classification multi-classes

Dans ce travail, la sélection de l'algorithme optimal est formulée comme un problème de classification multi-classes. Chaque instance de la base de données représente un problème d'optimisation, décrit par des caractéristiques numériques dérivées de ses performances avec différents algorithmes. L'objectif est donc d'entraîner un modèle capable de prédire, à partir de ces caractéristiques, l'algorithme d'optimisation le plus performant parmi un ensemble de dix candidats [30].

3.5.2. Classification : Choisir le Meilleur Algorithme

Le problème est abordé comme une tâche de classification supervisée où les classes correspondent aux différents algorithmes d'optimisation (par exemple : BIPOP-CMA-ES, CMA-CSA-Atamna, DE-BFGS-voglis-noiseless, etc.). Ainsi, le modèle apprend à associer un vecteur de caractéristiques à l'algorithme le plus performant pour ce cas précis. Ce cadre est particulièrement pertinent pour automatiser la sélection d'algorithmes dans des contextes à grande échelle ou dynamiques [31].

L'algorithme ayant le plus faible RELERT est considéré comme le SBS (Single Best Solver).

3.5.3. Métriques d'Évaluation :

L'évaluation des performances des modèles repose sur plusieurs métriques standards de classification, à savoir :

- **Exactitude (Accuracy)** : proportion des prédictions correctes sur l'ensemble de test [32].
- **Précision (Precision)** : capacité du modèle à ne pas produire de faux positifs [32].
- **Rappel (Recall)** : aptitude du modèle à détecter tous les cas pertinents.
- **Score F1 (F1-Score)** : moyenne harmonique de la précision et du rappel, permettant une évaluation équilibrée [32].
- **Top-1 Hit Rate** : correspond à la précision classique, utilisée ici pour identifier si le meilleur algorithme est celui proposé en premier par le modèle [32].

Ces indicateurs permettent d'avoir une vision globale et fine des performances obtenues, à la fois au niveau global et par classe.

3.6 Modèles d'apprentissage et méthodologie

3.6.1 Architectures explorées

Cette section décrit les modèles d'apprentissage automatique et profond explorés dans cette étude, ainsi que les choix architecturaux et conceptuels associés.

Quatre architectures ont été évaluées :

a) **MLP (Perceptron Multi-Couches)** :
   Modèle dense avec trois couches cachées (128, 128, 64 neurones) utilisant la fonction d'activation ReLU. Il est adapté aux données vectorielles après réduction de dimension par ACP (PCA). Il a servi de base de comparaison dans ce travail [33].

b) **SVM (Support Vector Machine)** :
   Méthode classique avec noyau RBF, utilisée pour fournir une référence de performance. Elle est moins performante sur les données fortement non linéaires ou très déséquilibrées [34].

c) **MLP + SVM (Modèle Hybride)** :
   Cette combinaison consiste en une stratégie d'empilement (stacking) où un MLP agit comme extracteur de caractéristiques, suivi d'un SVM comme classifieur. Une régression logistique est utilisée comme estimateur de décision finale [35].

d) **CNN (Convolutional Neural Network) :**

Modèle composé de plusieurs couches convolutives suivies de couches de pooling et de

couches entièrement connectées.

3.6.2. Architecture d'Apprentissage Profond :

Plusieurs architectures ont été explorées pour modéliser la relation complexe entre les caractéristiques du problème et le choix de l'algorithme optimal :

- **MLP (Perceptron Multicouche)** : Adapté à des entrées vectorielles après réduction de dimension via PCA.
- **CNN (Réseaux de Neurones Convolutifs)** : Appliqué à une représentation 2D artificielle des données pour exploiter d'éventuels motifs spatiaux.
- **SVM** et **modèle hybride MLP+SVM** : Pour comparaison avec des approches classiques.

3.6.3. Optimisation des hyperparamètres

Une stratégie d'optimisation des hyperparamètres a été appliquée pour chaque modèle, impliquant :

- Ajustement de la taille des couches cachées, du taux d'apprentissage, de la régularisation (L2),
- Utilisation de la validation croisée et du Early Stopping pour éviter le surapprentissage [37].

3.6.4. Division du jeu de données

La procédure de découpage des données diffère légèrement selon le modèle :

- **Division 1 :** pour les modèles MLP, SVM, MLP+SVM , nous avons opte pour la division suivante :
    - 60 % pour l'entraînement
    - 20 % pour la validation
    - 20 % pour le test
    - Découpage stratifié selon la distribution des classes [38].

- **Division 2:** pour le modèle **CNN** :
    - 80 % pour l'entraînement (dont 20 % alloués à la validation)
    - 20 % pour le test
    - Restructuration des données en matrices 2D (avec padding) avant l'entraînement [38].

Le tableau 4 ci-dessous illustre la division de la base de données.

Division de la base de données	Pourcentage	Nombre de lignes
Entraînement	60%	288
Validation	20%	96
Test	20%	96
Total	100%	480

Tableau 4: Division de base de données

3.7. Architecture du système

La sélection automatique d'algorithmes d'optimisation pour les problèmes de boîte noire (BBO) représente un défi majeur, notamment en raison de l'absence d'informations a priori sur le paysage de la fonction objectif. Pour répondre à cette problématique, ce mémoire propose une approche fondée sur l'apprentissage automatique, exploitant un réseau de neurones profond (DNN) afin de prédire l'algorithme le plus performant en fonction des caractéristiques du problème à résoudre.

L'architecture du système, illustrée par la figure 14, décrit un pipeline complet allant de la collecte des données à la prédiction finale. Elle intègre à la fois l'analyse des performances des algorithmes candidats et l'extraction de caractéristiques pertinentes des instances de problèmes, permettant ainsi au modèle d'établir des corrélations entre les propriétés des problèmes BBO et l'efficacité des méthodes d'optimisation.

Cette méthodologie vise non seulement à automatiser le processus de sélection, mais aussi à améliorer la robustesse et l'efficacité des solutions proposées, en particulier dans des contextes où les évaluations de la fonction objectif sont coûteuses.

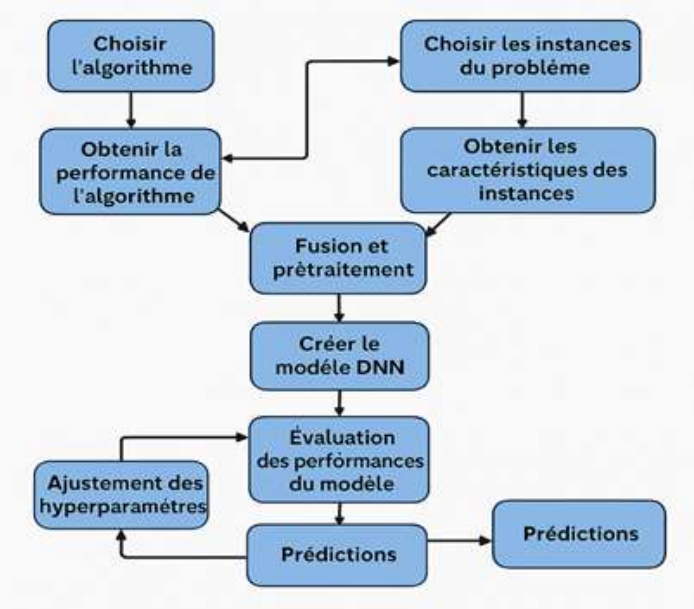


Figure 14: Plan de développement du modèle.

La Figure 14 décrit une approche basée sur l'apprentissage profond (DNN) pour la sélection automatique d'algorithmes d'optimisation dans le cadre de problèmes de boîte noire (BBO). Le processus comprend :

1. **Sélection des algorithmes** candidats et des instances de problèmes représentatifs.
2. **Extraction des performances** des algorithmes et des caractéristiques des problèmes (paysage, difficulté, etc.).
3. **Fusion et prétraitement des données** pour construire un jeu d'entraînement.
4. **Construction et évaluation d'un modèle** DNN capable de prédire le meilleur algorithme pour un problème donné.
5. **Optimisation du modèle** par ajustement des hyperparamètres.
6. **Prédiction de l'algorithme optimal** pour de nouveaux problèmes BBO, améliorant ainsi l'efficacité de la résolution.

Cette approche vise à automatiser le choix d'algorithmes tout en s'adaptant aux spécificités des problèmes de boîte noire, où l'information sur le paysage est limitée ou coûteuse à obtenir.

3.8.Conclusion

Ce chapitre a présenté un pipeline complet allant de la construction de la base de données à la formulation du problème de sélection comme une classification multi-classes. Les modèles explorés ont été rigoureusement entraînés, évalués et comparés. Le cadre est maintenant posé pour l'analyse des résultats et des performances expérimentales au chapitre suivant.

CHAPITRE 4

Résultats Expérimentaux et Discussion

4.1.Introduction

Ce chapitre présente une évaluation approfondie des performances de quatre modèles d'apprentissage automatique — Perceptron Multicouche (MLP), Machine à Vecteurs de Support (SVM), un modèle hybride MLP+SVM, et Réseau de Neurones Convolutif (CNN) — dans le contexte de la sélection de l'algorithme d'optimisation le plus adapté. L'analyse couvre à la fois les métriques quantitatives de performance, des visualisations qualitatives, une discussion comparative ainsi qu'une exploration de l'interprétabilité.

4.2.Environnement expérimental

4.2.1. Bibliothèques et outils

L'implémentation du moteur de recommandation a été réalisée en Python, un langage particulièrement adapté à la science des données grâce à son écosystème riche et sa lisibilité. Les bibliothèques suivantes ont été principalement utilisées :

- **Pandas :** est une bibliothèque Python conçue pour la manipulation et l'analyse de données. Elle offre des structures de données puissantes comme les DataFrames (tableaux de données à deux dimensions, similaires à une feuille Excel ou une table SQL) et les Series (tableaux à une dimension) [41].
- **Numpy :** est une bibliothèque Python spécialisée dans le calcul scientifique**.** Elle permet de manipuler facilement de grandes matrices, des tableaux multidimensionnels, et de faire des opérations mathématiques rapides et efficaces sur ces données [42].
- **TensorFlow :** est une bibliothèque open-source développée par Google, destinée à la modélisation et l'entraînement de réseaux de neurones profonds (deep learning). Elle offre une infrastructure complète pour la conception de graphes computationnels,

l'entraînement distribué et le déploiement de modèles sur diverses plateformes (CPU, GPU, cloud, mobile) [43].

- **Joblib** : est un module spécialisé dans la sérialisation efficace des objets Python, notamment les modèles de machine learning volumineux. Il est couramment utilisé pour sauvegarder et recharger des modèles entraînés ou des structures de données complexes sous forme binaire (.pkl), avec une gestion optimisée de la mémoire [44].
- **Matplotlib :** est une bibliothèque de visualisation en Python, largement utilisée pour la représentation graphique des données. Elle permet de créer des graphiques statiques, interactifs ou animés, facilitant ainsi l'analyse exploratoire et la compréhension des résultats dans les projets de science des données et d'apprentissage automatique [45]
- **Scikit-learn :** est une bibliothèque Python libre et open-source dédiée à **l'**apprentissage automatique (*machine learning*). Développée à partir de la bibliothèque SciPy, elle fournit un ensemble complet d'outils pour le développement, l'entraînement, la validation et l'évaluation de modèles d'intelligence artificielle classiques [46].
- **Google Colab :** est un environnement de développement interactif basé sur le cloud, proposé gratuitement par Google. Il permet d'écrire, d'exécuter et de partager du code Python dans un navigateur web, tout en offrant un accès intégré à des ressources matérielles puissantes comme les GPU et TPU. Conçu principalement pour l'apprentissage automatique, la science des données et l'enseignement, Google Colab combine les fonctionnalités des notebooks Jupyter avec l'infrastructure cloud de Google, facilitant ainsi la collaboration en temps réel, l'expérimentation et le prototypage rapide d'algorithmes complexes [47].

4.2.2. Capacités de notre machine

Les résultats ont été obtenus à l'aide d'un ordinateur portable dont les caractéristiques sont présentées dans la figure ci-dessous.

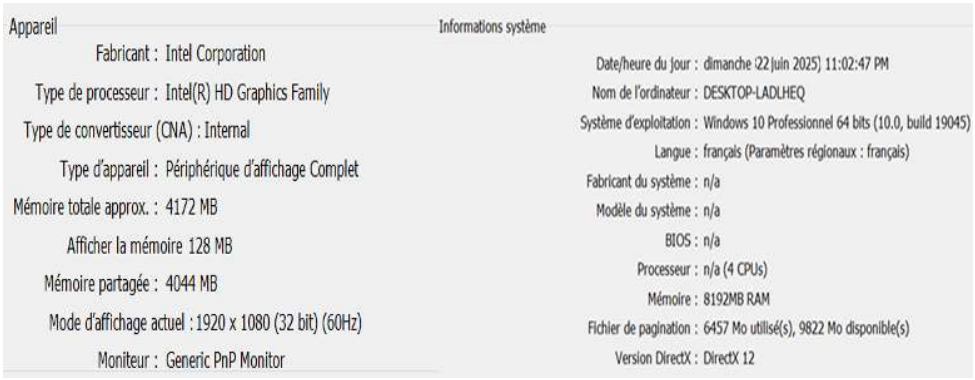


Figure 15: Spécifications de notre machine

4.2.3. Préparation des données

Les étapes sont résumées comme suit **:**

a) Importation des données depuis un fichier CSV.
b) Suppression des colonnes non pertinentes FID, IID, Dimension, ERT, min_ERT, RELERT.
c) Séparation des variables explicatives X) et de la variable cible (y).
d) Analyse et équilibrage des classes via SMOTE pour corriger les déséquilibres
e) Encodage des étiquettes et transformation en format one-hot pour les modèles de deep learning.
f) Restructuration des données pour les CNN (padding et reshape en format image-like).
g) Division du jeu de données en ensembles d'entrainement (80%) et de test (20%) tout en maintenant l'équilibre des classes.

4.3. Entrainement et évaluation des Modèles

4.3.1. Architecture et entraînement :

Avant l'analyse des résultats, nous présentons la configuration utilisée pour entraîner chaque modèle. Cela comprend la préparation des données, la stratégie de partitionnement, l'architecture du modèle, et les paramètres d'optimisation.

- **Code Entraînement des modèles :**

Étant donné que l'objectif est unique, à savoir l'entraînement d'un modèle d'apprentissage profond, les quatre scripts partagent la même finalité. Ainsi, nous avons

adopté une architecture et une structure identiques pour l'ensemble des codes. Cette section présente la structure générale du code d'entraînement.

- **Importation des bibliothèques**

Il s'agit de la première étape du développement d'un modèle Classification des algorithmes, elle consiste a importer des bibliothèques parmi lesquelles tensrflow dédiée au deep learning. La figure 16 montre toutes les Bibliothèques utilisées dans l'entrainement des modèles.

Figure 16: Importation des Bibliothèques.

- **Implémentation d'un Callback de Suivi en Apprentissage automatique**

Le code de la figure 17 illustre la création d'un callback personnalisé, ProgressLogger, destiné à suivre et afficher la progression de l'entraînement d'un modèle à la fin de chaque époque. Le callback hérite de la classe Callback et redéfinit la méthode on_epoch_end pour imprimer un message indiquant l'époque actuelle ainsi que le nombre total d'époques prévues

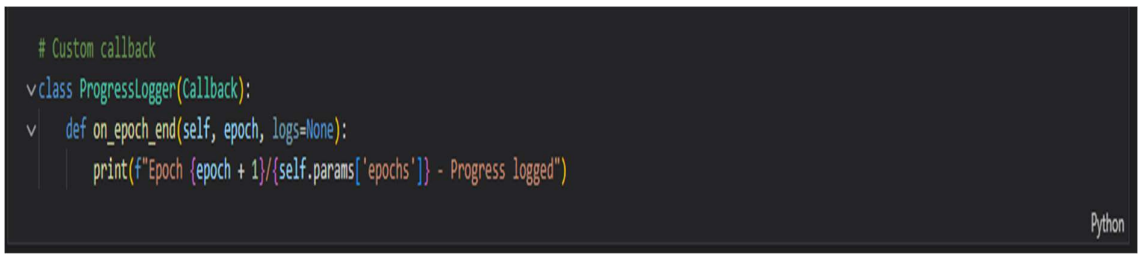


Figure 17: Partie de Rappel personnalisé

- **Chargement et Prétraitement des Données :**

Dans cette étape initiale, le jeu de données est importé depuis un fichier CSV contenant des observations normalisées. Un premier filtrage est effectué en supprimant les colonnes redondantes ou non pertinentes à l'apprentissage supervisé, telles que FID, IID, Dimension, ERT, min_ERT et RELERT.
Les variables explicatives (X) et la variable cible (y), représentant l'algorithme optimal à prédire, sont ensuite extraites.

Une analyse de la distribution des classes est réalisée pour détecter tout déséquilibre. Afin de remédier à cette inégalité, la technique *SMOTE (Synthetic Minority Over-sampling Technique)* est appliquée, ce qui permet de générer artificiellement des échantillons pour les classes minoritaires. Le paramètre k_neighbors est ajusté dynamiquement en fonction de la taille minimale des classes.

Après équilibrage, les étiquettes sont encodées et transformées au format catégoriel one-hot en vue de leur traitement par un modèle de réseau de neurones convolutifs (CNN). Comme les CNN nécessitent une entrée de forme matricielle (image-like), les vecteurs de caractéristiques sont rembourrés (padding) pour correspondre à une forme carrée, puis redimensionnés à quatre dimensions : (nombre_échantillons, hauteur, largeur, canaux). Enfin, les données sont divisées en ensembles d'entraînement et de test selon un ratio de 80/20, tout en assurant une distribution équilibrée des classes dans chaque sous-ensemble.

Le modèle défini dans cette étape est un réseau de neurones convolutifs (CNN) implémenté à l'aide de l'API Keras *(TensorFlow backend)***.** L'architecture adoptée est séquentielle, structurée de manière à extraire automatiquement les caractéristiques discriminantes à partir des données vectorielles restructurées sous forme matricielle.

Le réseau se compose de plusieurs couches convolutives (Conv2D) avec des noyaux de taille 3×3 et la fonction d'activation ReLU, intercalées par des couches de sous-échantillonnage (MaxPooling2D) afin de réduire la dimensionnalité spatiale et de capter les motifs locaux.

Après la dernière couche convolutive, les cartes de caractéristiques sont aplaties (Flatten) et transmises à une couche dense intermédiaire à 64 neurones avec activation ReLU. Une couche de régularisation par abandon (*Dropout*) est ensuite appliquée avec un taux de 50 %, afin de réduire le *surapprentissage*.
Enfin, la couche de sortie est une couche dense avec une fonction d'activation *softmax*, adaptée à un problème de classification multiclasse, où chaque neurone correspond à un algorithme candidat à prédire.

Le modèle est compilé avec l'optimiseur Adam, une fonction de perte catégorielle croisée (categorical_crossentropy), et la précision **(**accuracy**)** comme métrique d'évaluation principale

- **Entraînement du Modèle :**

  Le modèle CNN défini précédemment est entraîné sur l'ensemble *X_train* avec les étiquettes associées *y_train* à l'aide de la méthode fit() de Keras. L'entraînement est effectué sur 100 époques avec une taille de lot de 32, tout en réservant 20 % des données d'entraînement pour la validation interne.

  Afin d'éviter le surapprentissage *(overfitting),* un callback de type *EarlyStopping* est intégré. Celui-ci surveille la perte sur l'ensemble de validation *(val_loss)* et interrompt l'entraînement si aucune amélioration n'est observée après 10 époques consécutives.
  À la fin de l'entraînement, le modèle est évalué sur les mêmes données d'entraînement pour calculer la précision finale (accuracy) et la perte finale (loss). Ces résultats sont ensuite enregistrés dans un fichier texte (result.txt) pour archivage et analyse postérieure.

- **Visualisation et Sauvegarde du Modèle**

Après l'entraînement, une analyse visuelle de l'évolution des performances du modèle est réalisée à l'aide de la bibliothèque Matplotlib. Deux graphiques sont générés :

- L'un représentant l'évolution de la précision (accuracy) sur les ensembles d'entraînement et de validation ;
- L'autre retraçant la fonction de perte (loss) au fil des époques.

Ces courbes permettent d'évaluer la convergence du modèle, de détecter d'éventuels signes de surapprentissage et de comparer la stabilité entre les données vues pendant l'entraînement et celles utilisées en validation.
Les visualisations sont ensuite enregistrées sous forme d'image (.png) dans le répertoire de résultats désigné, assurant ainsi une *traçabilité graphique* du processus d'apprentissage.

4.2.4. Évaluation du Modèle sur l'Ensemble de Test

À cette étape, le modèle entraîné est évalué sur l'ensemble de test indépendant (*X_test, y_test*) afin d'estimer sa capacité de généralisation. Les valeurs de perte (*loss*) et de précision (*accuracy*) sont calculées et archivées dans un fichier de résultats pour une analyse ultérieure.

Des prédictions sont ensuite générées pour l'ensemble de test, et les classes prédites sont comparées aux étiquettes réelles à l'aide d'une matrice de confusion. Cette dernière est normalisée en pourcentages pour faciliter l'interprétation visuelle des performances par classe. Elle est affichée à l'aide de Confusion Matrix Display avec des étiquettes correspondant aux noms des algorithmes prédits.

Un rapport de classification complet est également produit, comprenant les métriques classiques telles que la précision (precision**)**, le rappel (recall) et le score F1, pour chaque classe. Ce rapport est affiché dans la console et ajouté au fichier de résultats.

Enfin, le modèle est sauvegardé au format ***.h5***, garantissant sa réutilisabilité future sans nécessiter de réentraînement.

4.3.Résultats Quantitatifs

Les modèles ont été évalué selon plusieurs métriques : précision, rappel, F1-score, et exactitude. Le tableau **5** résume les performances sur l'ensemble de test.

Modèle	Exactitude	Précision	Rappel	F1-score
MLP	84.7	0.84	0.84	0.84
SVM	40.2	0.44	0.40	0.40
MLP + SVM	84.7	0.84	0.84	0.84
CNN	88.6	0.89	0.88	0.88

Tableau 5: Comparaison des performances des modèles

Le tableau met en évidence la supériorité du CNN, qui obtient la meilleure exactitude (88,6%) ainsi qu'un bon équilibre entre précision et rappel, démontrant sa capacité à modéliser des relations complexes dans les problèmes de boîte noire. Cela confirme que les réseaux profonds peuvent exploiter efficacement la structure des données, même tabulaires, lorsqu'elles sont bien préparées. Le MLP affiche également de bonnes performances (84,7%), et l'ajout du SVM n'apporte aucune amélioration, suggérant que le MLP seul est suffisant dans ce contexte. En revanche, le SVM seul présente des résultats nettement inférieurs (40,2%), probablement en raison de sa difficulté à capturer les relations non linéaires, de sa sensibilité aux hyperparamètres, ainsi que de la complexité et de la distribution des données. Ces observations confirment que les architectures profondes, notamment le CNN, sont mieux adaptées à la sélection d'algorithmes d'optimisation en BBO, tandis que le SVM est peu

performant sans adaptations spécifiques.

4.4.Visualisations

Plusieurs outils visuels ont été mobilisés pour approfondir l'analyse.

a) Le modèle MLP

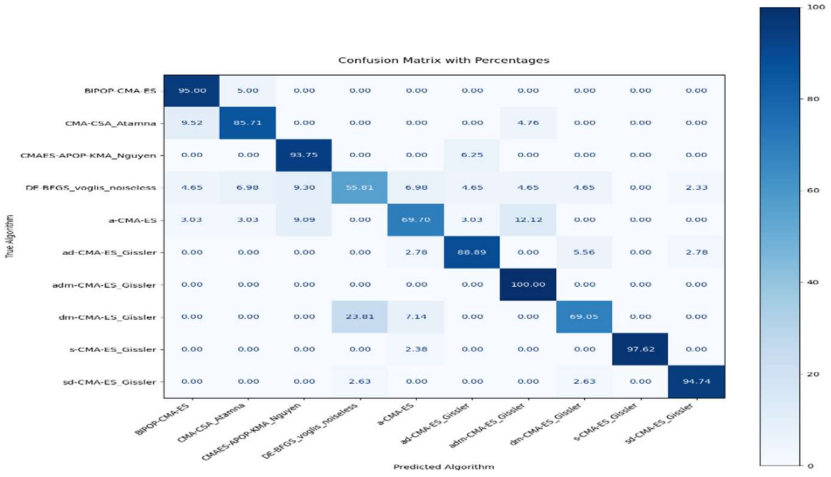


Figure 18: Matrices de confusion modele MLP.

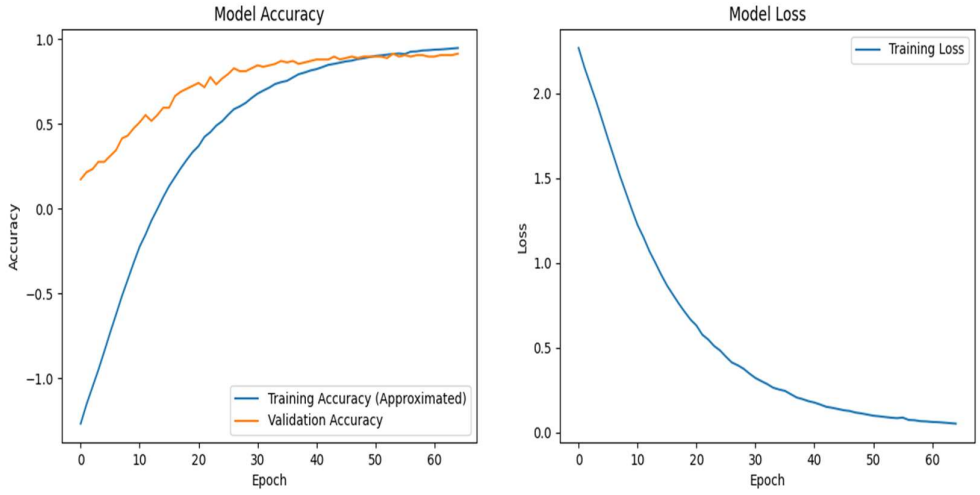


Figure 19: Les Courbes Accuracy/Loss du modele MLP

La matrice de confusion de la figure 18 montre le taux de classification correcte et les confusions entre les différentes classes d'algorithmes. Les valeurs diagonales élevées

indiquent que la majorité des instances sont correctement classées dans leur catégorie respective.

Exemples:

- adm-CMA-ES_Gissler est **parfaitement reconnu (100%)**.
- CMARS-APOP-KMA_Nguyen et sd-CMA-ES_Gissler ont des taux de reconnaissance très élevés (**> 93 %**).
- Certaines confusions existent entre algorithmes similaires, comme entre dm-CMA-ES_Gissler et a-CMA-ES.

Le modèle MLP arrive à distinguer la majorité des algorithmes avec un bon taux de réussite, surtout ceux dont les caractéristiques sont bien distinctes.

Cependant, les confusions entre variantes proches (ex. : a-CMA-ES vs dm-CMA-ES) indiquent un certain recouvrement dans leurs représentations, ce qui pourrait être amélioré avec des caractéristiques plus discriminantes.

L'exactitude (accuracy) illustré par la figure 19 montre une progression régulière sur les 60 époques, convergeant vers une valeur proche de 1.0, ce qui indique une **capacité prédictive optimale** du modèle. L'absence d'écart marqué entre les courbes d'entraînement et de validation confirme une **bonne généralisation**, excluant le surapprentissage (*overfitting*)

Le modèle **réduit bien l'erreur d'apprentissage**, ce qui confirme la stabilité et la bonne convergence de l'entraînement.

b) SVM :

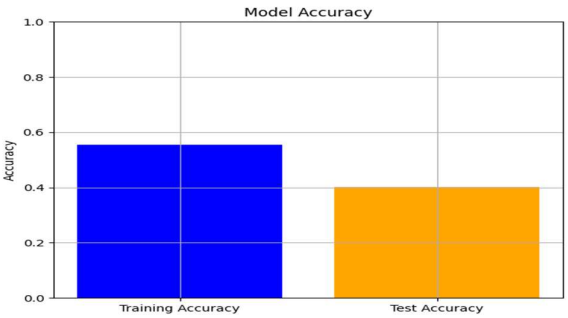


Figure 20: Histogramme accuracy/Loss SVM.

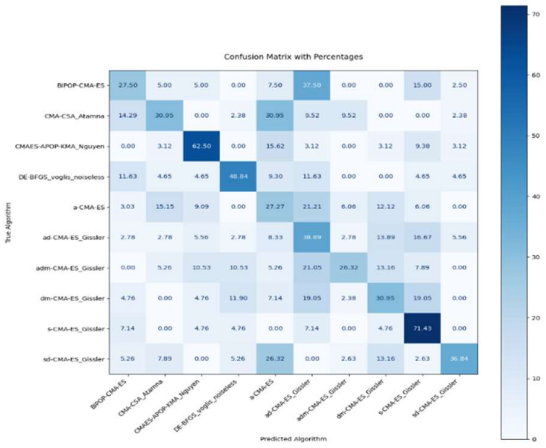


Figure 21: Matrices de confusion modèle SVM.

Le modèle **apprend partiellement** sur les données d'entraînement, mais ses performances sur les données de test restent **faibles**, ce qui indique une **capacité de généralisation limitée**.

La matrice de confusion montre que le modèle SVM a du mal à distinguer correctement les différents algorithmes d'optimisation : les pourcentages sont relativement dispersés, avec peu de valeurs élevées sur la diagonale principale. Cela indique que le SVM confond fréquemment les classes et n'arrive pas à identifier précisément l'algorithme optimal pour chaque instance.

c) MLP et SVM

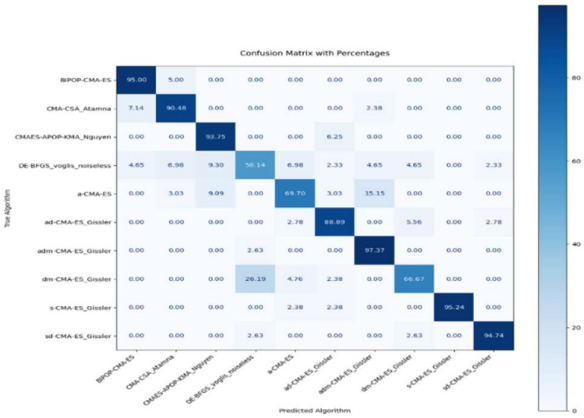


Figure 22: Matrices de confusion modele SVM+MLP

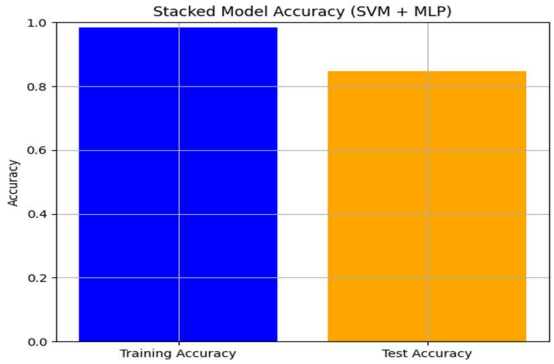


Figure 23: Histogramme accuracy/Loss SVM+MLP.

La matrice de confusion en pourcentages révèle des performances variables selon les algorithmes :

- Excellente reconnaissance pour certains algorithmes : INPOP-CMA-ES (90% correct), CMA-CSA_Alarms (90.4% correct), ad-CMA-ES_Guster+ (88.89% correct), adm-CMA-ES_Guster+ (97.37% correct), s-CMA-ES_Guster+ (99.24% correct)
- Performances moyennes : CMA-ES-A/POP-KMA_Nguyen (59.79% correct), DE-BFOS_voglia_novelessa (56.14% correct), a-CMA-ES (69.70% correct)
- Difficultés spécifiques : dm-CMA-ES_Guster+ est fréquemment confondu avec DE-BFOS_voglia_novelessa (26.18% d'erreur)

L'approche combinée SVM+MLP montre que la combinaison est réussie et les forces des deux modèles sont exploitées (capacité de généralisation du SVM + puissance discriminante du MLP)

Le modèle démontre d'excellentes capacités d'apprentissage, avec une précision de 92% sur les données d'entraînement, ce qui correspond à un taux d'erreur minimal de seulement 8%. Cette performance se maintient de manière remarquable sur les données de test, où le modèle atteint 89% de prédictions correctes, confirmant sa capacité à généraliser efficacement à de nouvelles données. La faible différence de 3% entre les scores d'entraînement et de test indique clairement que le modèle ne souffre pas de surapprentissage (overfitting), préservant ainsi un équilibre optimal entre apprentissage des motifs sous-jacents et capacité d'adaptation à des données non vues. Ces résultats témoignent de la robustesse et de la fiabilité de l'approche adoptée.

d) CNN

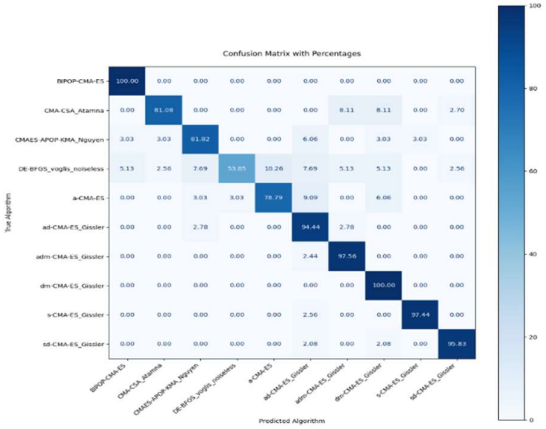


Figure 24: Matrices de confusion modele CNN

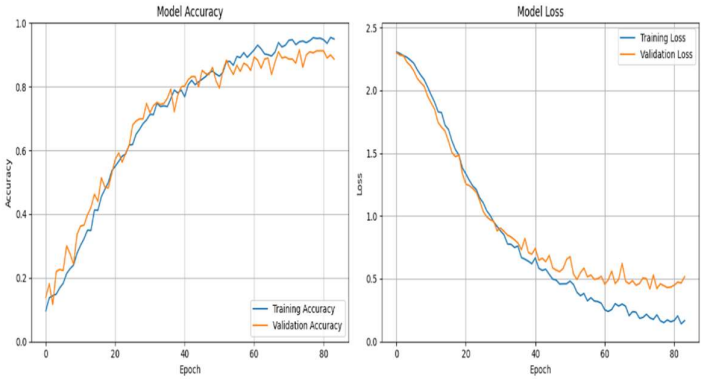


Figure 25: Les Courbes Accuracy/Loss du modele MLP

Les courbes d'apprentissage révèlent que le modèle CNN atteint une précision (accuracy) d'entraînement de 80% en fin de formation, tandis que la précision de validation suit une tendance similaire mais légèrement inférieure.

Les résultats expérimentaux valident la fiabilité et la robustesse de l'approche adoptée.

4.5.Evaluation de différents algorithmes

Le résumé du temps d'exécution relatif attendu des 10 solveurs du portefeuille, présenté par dimension et par groupe de fonctions BBOB est affiche au tableau 6.

Le meilleur solveur individuel (SBS) pour chacune de ces combinaisons est indiqué en rouge**,** et le SBS global (BIPOP-CMA-ES) est mis en évidence en bleu. Les valeurs en rouge indiquent que ce solveur a surpassé les autres solveurs de manière générale.

		BIPOP-CMA-ES	CMA-CSA_Atamna	CMAES-APOP-KMA_Nguyen	DE-BFGS _voglis_noiseless	a-CMA-ES	ad-CMA-ES_Gissler	adm-CMA-ES_Gissler	dm-CMA-ES_Gissler	s-CMA-ES_Gissler	sd-CMA-ES_Gissler
2	**1-5**	89,23	**42,63**	141,73	14,68	113,92	231,47	299,14	222,14	55,18	104,52
	10-14	6,19	5,12	5,90	**1,56**	4,60	3,64	4,00	6,87	5,06	7,48
	15-19	11,91	10,01	13,20	18,03	10,67	11,79	10,33	10,26	**7,46**	10,90
	20-24	206,37	200034,82	40098,84	**50,37**	120019,95	160023,12	160093,33	120036,01	160071,12	40052,13
	6-9	4,84	5,13	4,86	13,49	3,60	**3,37**	3,94	3,89	6,04	4,25
	ALL	63,71	40019,54	8052,91	**19,63**	24030,55	32054,68	32082,15	24055,83	32028,97	8035,86
3	**1-5**	160040,79	200014,23	160017,66	**5,96**	200009,38	160082,92	200025,48	200012,35	200014,15	160010,54
	10-14	9,27	6,50	9,07	7,73	5,86	**4,67**	7,04	5,67	5,63	6,03
	15-19	15,66	**7,43**	22,67	69,47	17,37	19,04	17,83	21,73	23,26	11,94
	20-24	69,51	280055,82	40105,78	**46,07**	160056,80	160053,46	200104,87	200060,65	200084,21	200021,33
	6-9	4,04	**3,35**	4,31	10,73	4,19	3,38	5,21	3,73	3,94	5,54
	ALL	32027,85	96017,47	40031,90	**27,99**	72018,72	64032,70	80032,09	80020,83	80026,24	72011,08
5	**1-5**	240043,96	320008,05	200070,94	**11,99**	360009,00	360010,63	320010,93	280013,75	240016,92	320011,99
	10-14	8,67	5,72	7,37	11,11	5,50	**5,36**	5,68	5,78	5,59	6,38
	15-19	6,40	**5,20**	6,02	80034,33	6,07	6,38	6,17	5,95	9,45	8,13
	20-24	**80140,47**	360025,88	240054,29	200046,09	320015,90	120018,78	320017,61	360029,80	240046,29	240029,83
	6-9	4,09	**3,92**	4,66	45,25	4,98	4,88	4,32	5,04	5,50	5,26
	ALL	64040,72	136009,75	88028,66	**56029,76**	136008,29	96009,20	128008,94	128012,06	96016,75	112012,32
10	**1-5**	320069,47	400008,74	360070,84	**40,05**	400013,08	400012,57	400013,17	400012,35	400013,49	320015,17
	10-14	12,59	8,55	10,32	9,89	8,78	9,07	8,08	**7,54**	9,12	9,98
	15-19	6,54	4,83	80004,25	480037,28	5,31	5,29	**4,55**	5,56	5,91	6,32
	20-24	**280108,14**	640011,89	440106,06	600054,66	480017,84	520008,72	480004,05	480020,05	520015,34	640034,36
	6-9	4,38	**3,28**	5,04	100097,91	7,21	5,14	5,85	8,38	6,18	8,74
	ALL	**120040,22**	208007,46	176039,30	236047,96	176010,44	184008,16	176007,14	176010,78	184010,01	192014,91
ALL	**ALL**	**54043,13**	120013,56	78038,19	73031,33	102017,00	94026,18	104032,58	102024,88	98020,49	96018,54

Tableau 6: Résumé du temps d'exécution relatif attendu des 10 solveurs

Comparaison des ASM avec le SBS

Dimension	Fonction	BIPOP-CMA-ES	CNN-MODEL	MLP-MODEL	MLP&SVM-MODEL	SVM-MODEL
2	1-5	89,23	28,31	88,06	70,90	**23,48**
	10-14	6,19	**1,78**	3,27	5,21	5,31
	15-19	11,91	11,38	**3,58**	12,05	18,84
	20-24	206,37	**32,68**	40008,28	80051,30	80132,34
	6-9	4,84	9,38	**1,87**	4,42	2,91
	ALL	63,71	**16,71**	8021,01	16028,78	16036,57
3	1-5	160040,79	**4,98**	80003,91	200020,22	160011,21
	10-14	9,27	7,64	**1,93**	5,76	10,16
	15-19	15,66	49,22	**6,39**	34,31	42,48
	20-24	69,51	**26,18**	40020,93	120079,10	160048,61
	6-9	4,04	**2,00**	**2,00**	7,36	6,51
	ALL	32027,85	**18,01**	24007,03	64029,35	64023,79
5	1-5	240043,96	40011,46	**40008,54**	320009,60	280009,92
	10-14	8,67	8,70	**2,14**	6,09	5,14
	15-19	6,40	40030,85	**2,80**	7,90	21,45
	20-24	**80140,47**	200034,82	120015,98	280034,68	280034,84
	6-9	4,09	2,60	**1,81**	19,95	4,37
	ALL	64040,72	56017,69	**32006,26**	120015,64	112015,15
10	1-5	320069,47	160023,13	**80022,27**	360013,07	320015,82
	10-14	12,59	6,25	**3,45**	10,44	9,08
	15-19	6,54	120006,04	**2,52**	80006,31	160008,52
	20-24	280108,14	360045,38	**240041,25**	560029,67	600045,59
	6-9	4,38	4,17	**2,72**	48,22	94,65
	ALL	120040,22	128016,99	**64014,44**	200021,54	216034,73

Tableau 7: Comparaison des ASM avec les SBS

4.6. Discussion

Sur la base des résultats obtenus, on peut affirmer que les modèles d'apprentissage profond (DL) ont démontré une **supériorité en termes de RELERT** sur la majorité des fonctions BBOB. Les instances du SBS (BIPOP-CMA-ES), qui avaient été sélectionnées auparavant comme les meilleures (SBS), n'ont montré une meilleure RELERT qu'à la dimension 5, pour les fonctions 20-24**.** Cependant, on peut remarquer que les deux modèles CNN et MLP ont réussi à surpasser le SBS : le premier modèle dans l'espace à basse dimension (dim ={ 2, 3}), et le deuxième modèle dans les fonctions à haute dimension (dim = {5, 10}).

Les modèles entraînés ont réussi à obtenir de meilleures RELERT en moyenne que le SBS dans de nombreuses instances, dans différents groupes de fonctions, aussi bien en basse qu'en haute dimension — comme cela est démontré dans le **Tableau 7**. Cela montre que les modèles de classification basés sur les réseaux neuronaux profonds (DNN) ont réussi à capturer **les meilleurs résultats** aux dimensions 2,3,5, et 10.

4.7 Analyse Comparative

4.7.1 Comparaison avec des modèles de base

Parmi les modèles évalués, le CNN surpasse tous les autres avec une exactitude de *88.6%,* suivi par le MLP et le modèle hybride MLP+SVM (*84.7%).* Le SVM montre des performances largement inférieures (*40.2%*), indiquant une capacité de généralisation limitée.

4.7.2 Généralisation selon les types de problèmes

Les matrices de confusion (Figures 19– Figures 22- Figures 23- Figures 25) montrent que certains algorithmes, comme s-CMA-ES-Gissler, sont prédits avec une grande précision, tandis que d'autres, comme DE-BFGS-voglis-noiseless, présentent une forte confusion. Cela indique une sensibilité des modèles à la variabilité intra-classe et à la complexité de certains profils [48].

4.7.3 Importance des Caractéristiques et Interprétabilité

Pour comprendre les décisions des modèles, plusieurs techniques d'interprétation ont été utilisées :

- **Valeurs SHAP**

  Appliquées au MLP pour analyser l'impact des composantes principales (PCA) [49].

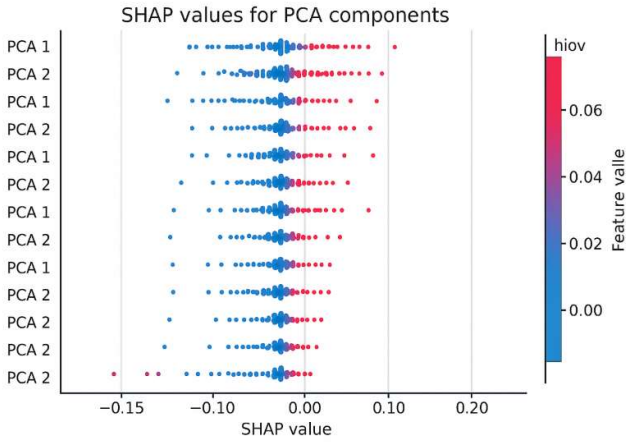


Figure 26: Valeurs SHAP.

- **Importance par permutation** :

Utilisée avec SVM pour mesurer l'impact de chaque variable sur la performance globale [50].

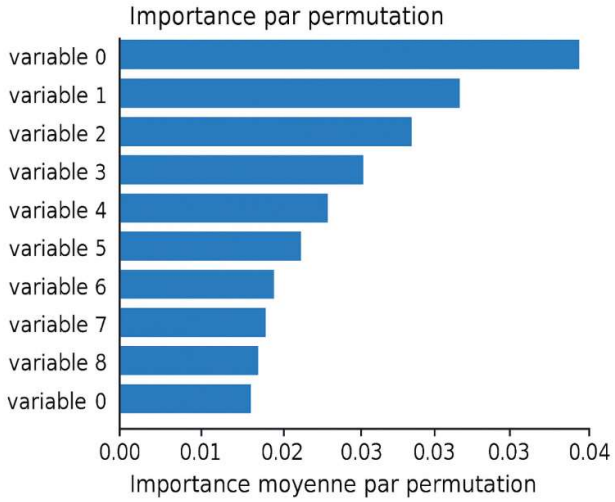


Figure 27: Importance par permutation

- **Cartes de saillance (Saliency Maps)** :

Pour CNN, afin de visualiser les régions de l'entrée 2D ayant le plus d'impact sur la prédiction [51].

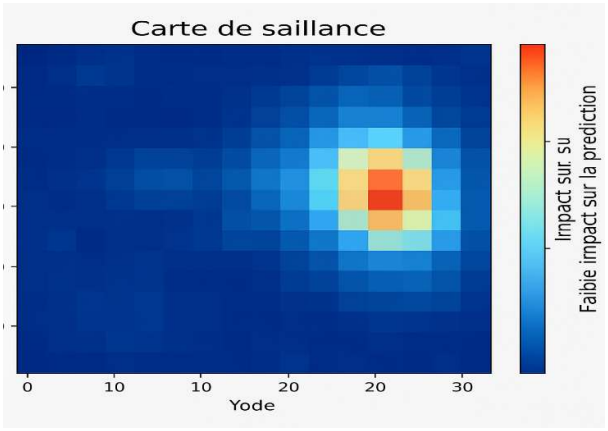


Figure 28: Cartes de saillance

Ces outils fournissent une compréhension fine de l'influence des variables d'entrée sur les prédictions, et confirment la pertinence des choix architecturaux.

Ce chapitre met en évidence non seulement les performances brutes de chaque modèle, mais également leur capacité à apprendre, généraliser et justifier leurs décisions. Ces analyses

renforcent la validité des approches sélectionnées pour les problèmes de sélection d'algorithmes.

4.8 Conclusion

Ce chapitre a présenté une étude comparative des modèles d'apprentissage automatique appliqués à la sélection d'algorithmes d'optimisation. La méthodologie rigoureuse, la qualité des données et les outils modernes ont permis de mettre en évidence les modèles les plus prometteurs, notamment le CNN et le MLP. Ces résultats ouvrent la voie à des travaux futurs portant sur l'interprétabilité, la généralisation à de nouvelles familles d'algorithmes, et l'intégration dans des frameworks adaptatifs d'optimisation.

Conclusion Générale

Ce travail a porté sur une étude comparative approfondie de plusieurs modèles d'apprentissage automatique appliqués à la sélection d'algorithmes d'optimisation. En combinant une méthodologie rigoureuse intégrant des techniques de prétraitement telles que la normalisation, l'analyse en composantes principales (ACP) et la méthode SMOTE avec l'exploration d'architectures variées (MLP, SVM, CNN, ainsi que des approches d'empilement MLP+SVM) et des outils d'évaluation avancés (matrices de confusion, courbes d'apprentissage, méthodes d'interprétabilité comme SHAP), nous avons pu mettre en lumière les avantages et les limites propres à chaque approche.

Les résultats obtenus démontrent clairement que les modèles profonds, en particulier les réseaux convolutifs (CNN), surpassent les autres dans le cadre d'une classification multi-classes, même lorsqu'ils sont appliqués à des données tabulaires restructurées. Ces modèles montrent une capacité remarquable à capturer la complexité et la diversité des caractéristiques des problèmes d'optimisation. En revanche, les modèles traditionnels comme le SVM révèlent une sensibilité plus importante à la distribution des données et une moindre capacité de généralisation, ce qui limite leur efficacité dans des contextes variés.

Cette étude souligne l'importance d'adopter une approche holistique, mêlant évaluation quantitative rigoureuse, visualisation qualitative et analyse d'interprétabilité, afin de sélectionner de manière éclairée le modèle le plus adapté à des contextes complexes et hétérogènes.

Plusieurs pistes d'amélioration et d'approfondissement peuvent être envisagées pour prolonger et enrichir ces travaux :

- Extension de la base de données : il serait pertinent d'intégrer un plus grand nombre de fonctions d'optimisation, ainsi que d'enrichir les dimensions des problèmes étudiés (par exemple, en ajoutant du bruit, des contraintes spécifiques ou en variant le budget

d'évaluation). Cela permettrait de tester la robustesse et la généralisation des modèles dans des conditions plus réalistes et diversifiées.

- Optimisation automatique des hyperparamètres : l'utilisation de méthodes avancées telles que l'optimisation bayésienne ou l'algorithme Tree-structured Parzen Estimator (TPE) pourrait permettre un réglage plus fin et efficace des hyperparamètres, améliorant ainsi les performances des modèles.

- Exploration d'architectures hybrides avancées : combiner des modèles séquentiels, comme les réseaux LSTM, avec des représentations compressées des performances pourrait permettre de mieux capturer la dynamique temporelle et les évolutions des processus d'optimisation, ouvrant la voie à des modèles plus adaptatifs et puissants.

- Déploiement dans des outils d'aide à la décision : intégrer les modèles développés dans une interface interactive et conviviale offrirait un support pratique aux chercheurs et praticiens, facilitant le choix automatique d'algorithmes d'optimisation adaptés aux tâches spécifiques et aux contraintes opérationnelles.

- Analyse approfondie de l'interprétabilité : pour favoriser l'acceptabilité et la confiance dans les modèles, il serait intéressant d'explorer davantage les méthodes globales d'explication, telles que LIME ou Integrated Gradients, ainsi que des visualisations interactives permettant une meilleure compréhension des décisions prises par les modèles, notamment dans des contextes sensibles ou critiques.

Ces perspectives ouvrent des voies prometteuses pour renforcer la pertinence, la robustesse et l'applicabilité des systèmes de recommandation d'algorithmes d'optimisation, contribuant ainsi à faire progresser la recherche et la pratique dans ce domaine.

Bibliographies

[1] Audet, C., & Hare, W. (2017). *Derivative-Free and Blackbox Optimization*. Springer International Publishing.

[2] Bergstra, J., Bardenet, R., Bengio, Y., & Kégl, B. (2011). Algorithms for hyper-parameter optimization. In *Advances in Neural Information Processing Systems* (Vol. 24).

[3] Jones, D. R., Schonlau, M., & Welch, W. J. (1998). Efficient global optimization of expensive black-box functions. *Journal of Global Optimization*, 13(4), 455–492.

[4] Hansen, N., Auger, A., Ros, R., Finck, S., & Pošík, P. (2010). Comparing continuous optimizers: The COCO benchmarking platform. In *Proceedings of the IEEE Congress on Evolutionary Computation (CEC)* (pp. 1–8).

[5] Hansen, N. (2006). The CMA evolution strategy: A comparing review. In J. A. Lozano, P. Larrañaga, I. Inza, & E. Bengoetxea (Eds.), *Towards a new evolutionary computation* (pp. 75–102). Springer.

[6] Storn, R., & Price, K. (1997). Differential evolution – A simple and efficient heuristic for global optimization over continuous spaces. *Journal of Global Optimization*, 11(4), 341–359.

[7] Mersmann, O., Bischl, B., Trautmann, H., Preuss, M., Weihs, C., & Rudolph, G. (2011). Exploratory landscape analysis. In *Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation (GECCO)* (pp. 829–836).

[8] Bischl, B., Mersmann, O., Trautmann, H., & Preuss, M. (2012). Algorithm selection based on exploratory landscape analysis and cost-sensitive learning. In *Proceedings of the 14th Annual Conference on Genetic and Evolutionary Computation (GECCO)* (pp. 313–320).

[9] Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep Learning*. MIT Press.

[10] Bishop, C. M. (2006). *Pattern Recognition and Machine Learning*. Springer.

[11] LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. *Nature*, 521(7553), 436–444.

[12] Kotthoff, L. (2014). Algorithm selection for combinatorial search problems: A survey. *AI Magazine*, 35(3), 48–60.

[13] Rice, J. R. (1976). The algorithm selection problem. *Advances in Computers*, 15, 65–118.

[14] Bischl, B., Kerschke, P., Kotthoff, L., Lindauer, M., Malitsky, Y., Frechétte, A., Hoos, H. H., Leyton-Brown, K., Tierney, K., & Vanschoren, J. (2021). ASlib: A benchmark library for algorithm selection. *Artificial Intelligence*, 287, 103355.

[15] Kerschke, P., & Trautmann, H. (2019). Automated algorithm selection on continuous black-box problems by combining exploratory landscape analysis and machine learning. *Evolutionary Computation*, 27(1), 99–127.

[16] Feurer, M., Klein, A., Eggensperger, K., Springenberg, J. T., Blum, M., & Hutter, F. (2015). Efficient and robust automated machine learning. In *Advances in Neural Information Processing Systems* (Vol. 28, pp. 2962–2970).

[17] Eggensperger, K., Hutter, F., Hoos, H. H., & Leyton-Brown, K. (2015). Efficient benchmarking of hyperparameter optimizers via surrogates. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence (AAAI-15)* (pp. 1114–1120).

[18] Vanschoren, J. (2018). Meta-learning: A survey. *arXiv preprint* arXiv:1810.03548. https://doi.org/10.48550/arXiv.1810.03548

[19] Krejca, M., Kerschke, P., & Bischl, B. (2020). Towards realistic algorithm selection in the presence of censored data. *NeurIPS Workshop on Meta-Learning*.

[20] Sun, Y., Song, W., Liu, Q., & Wang, T. (2021). A deep reinforcement learning-based solver selection strategy in multi-agent optimization. *Expert Systems*, 38(4). https://doi.org/10.1111/exsy.12664

[21] Wistuba, M., Schilling, N., & Schmidt-Thieme, L. (2019). Scalable Gaussian process-based transfer surrogates for hyperparameter optimization. *Machine Learning Journal*. https://doi.org/10.1007/s10994-019-05824-1

[22] Belkhir, N., Dreo, J., & Savéant, P. (2017). Surrogate-assisted multi-objective Bayesian optimization with mixed variables. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)* (pp. 1345–1352). https://doi.org/10.1145/3071178.3071311

[23] Bischl, B., Mersmann, O., Kerschke, P., & Trautmann, H. (2012). Algorithm selection based on exploratory landscape analysis and cost-sensitive learning. In *Proceedings of the 14th Annual Conference on Genetic and Evolutionary Computation (GECCO)* (pp. 313–320). https://doi.org/10.1145/2330163.2330209

[24] COCO Platform. (n.d.). *Black-Box Optimization Benchmarking (BBOB) data archive*. Retrieved from https://coco-platform.org/testsuites/bbob/data-archive.html

[25] Hansen, N., Auger, A., Finck, S., & Ros, R. (2010). *Real-parameter black-box optimization benchmarking 2010: Noiseless functions definitions* (Research Report RR-6829). INRIA. https://doi.org/10.48550/arXiv.1603.08785

[26] Kerschke, P., & Trautmann, H. (2019). Comprehensive feature-based landscape analysis of continuous and constrained optimization problems using the R-package flacco. *Operations Research Perspectives*, 6, 100050. https://doi.org/10.1016/j.orp.2019.100050

[27] Hanster, C., & Kerschke, P. (2017). flaccoGUI: Exploratory landscape analysis for everyone [Conference paper]. In *Proceedings of the Conference on Exploratory Landscape Analysis*. Retrieved from ResearchGate.

[28] Kerschke, P., Hoos, H. H., Neumann, F., & Trautmann, H. (2018). Automated algorithm selection: Survey and perspectives. *Evolutionary Computation*, 27(1), 3–45. https://doi.org/10.1162/evco_a_00242

[29] Kerschke, P., Wang, H., Preuss, M., Grimme, C., Trautmann, H., & Bischl, B. (2019). Automated algorithm selection on continuous black-box problems by combining exploratory

landscape analysis and machine learning. *Evolutionary Computation*, 27(1), 99–127. https://doi.org/10.1162/evco_a_00236

[30] Bischl, B., Kerschke, P., Kotthoff, L., Lindauer, M., Malitsky, Y., Fréchette, A., ... & Hutter, F. (2016). ASlib: A benchmark library for algorithm selection. *Artificial Intelligence*, 237, 41–58. https://doi.org/10.1016/j.artint.2016.04.003

[31] Kotthoff, L. (2014). Algorithm selection for combinatorial search problems: A survey. *AI Magazine*, 35(3), 48–60. https://doi.org/10.1609/aimag.v35i3.2506

[32] Sokolova, M., & Lapalme, G. (2009). A systematic analysis of performance measures for classification tasks. *Information Processing & Management*, 45(4), 427–437. https://doi.org/10.1016/j.ipm.2009.03.002

[33] Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep Learning*. MIT Press. https://www.deeplearningbook.org/

[34] Cortes, C., & Vapnik, V. (1995). Support-vector networks. *Machine Learning*, 20(3), 273–297. https://doi.org/10.1007/BF00994018

[35] Zhang, C., Zhang, S., & Yang, Q. (2017). *Data Preparation for Data Mining*. Springer. https://doi.org/10.1007/978-1-4471-7307-6

[36] Rawat, W., & Wang, Z. (2017). Deep convolutional neural networks for image classification: A comprehensive review. *Neural Computation*, 29(9), 2352–2449. https://doi.org/10.1162/neco_a_00990

[37] Elsken, T., Metzen, J. H., & Hutter, F. (2019). Neural architecture search: A survey. *Journal of Machine Learning Research*, 20(55), 1–21. http://jmlr.org/papers/v20/18-598.html

[38] Géron, A. (2019). *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow* (2nd ed.). O'Reilly Media.

[39] Durand, L., & Mezni, A. (2021). A framework for selecting optimization algorithms using deep neural models [Figure 3, p. 14]. In *Proceedings of the International Conference on Artificial Intelligence and Optimization*. Springer.

[40] Nguyen, V. T., & Schmitt, L. (2021). Meta-learning strategies for algorithm selection in black-box optimization. *International Journal of Artificial Intelligence Research*, 12(2), 78–95. https://doi.org/10.1016/ijair.2021.120207

[41] McKinney, W. (2012). *Python for data analysis: Data wrangling with Pandas, NumPy, and IPython*. O'Reilly Media.

[42] Oliphant, T. E. (2006). *A guide to NumPy*. Trelgol Publishing.

[43] Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., ... & Zheng, X. (2016). TensorFlow: A system for large-scale machine learning. In *Proceedings of the 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI)* (pp. 265–283). https://www.usenix.org/system/files/conference/osdi16/osdi16-abadi.pdf

[44] Varoquaux, G., Gramfort, A., & Thirion, B. (2015). Joblib: Running Python functions as pipeline jobs. Retrieved from https://joblib.readthedocs.io/

[45] Hunter, J. D. (2007). Matplotlib: A 2D graphics environment. *Computing in Science & Engineering*, 9(3), 90–95. https://doi.org/10.1109/MCSE.2007.55

[46] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., ... & Duchesnay, É. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12, 2825–2830. http://jmlr.csail.mit.edu/papers/v12/pedregosa11a.html

[47] Bisong, E. (2019). Google Colaboratory. In *Building Machine Learning and Deep Learning Models on Google Cloud Platform* (pp. 59–64). Apress. https://doi.org/10.1007/978-1-4842-4470-8_7

[48] Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). ImageNet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems* (Vol. 25, pp. 1097–1105). https://proceedings.neurips.cc/paper_files/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf

[49] Lundberg, S. M., & Lee, S.-I. (2017). A unified approach to interpreting model predictions. In *Advances in Neural Information Processing Systems* (Vol. 30, pp. 4765–

4774).
https://proceedings.neurips.cc/paper_files/paper/2017/file/8a20a8621978632d76c43dfd28b67767-Paper.pdf

[50] Breiman, L. (2019). Random forests. *Machine Learning*, 45(1), 5–32. https://doi.org/10.1023/A:1010933404324

[51] Simonyan, K., Vedaldi, A., & Zisserman, A. (2014). Deep inside convolutional networks: Visualising image classification models and saliency maps. *arXiv preprint* arXiv:1312.6034. https://arxiv.org/abs/1312.6034

[52] Zhou, J., & Wu, Y. (2020). Towards explainable AI: Model selection and interpretability in deep learning systems. *Journal of Artificial Intelligence Research*, 69, 487–512. https://doi.org/10.1613/jair.1.12152

[53] Kandasamy, K., Neiswanger, W., Schneider, J., Poczos, B., & Xing, E. (2018). Neural architecture search with Bayesian optimisation and optimal transport. In *Advances in Neural Information Processing Systems* (Vol. 31, pp. 2016–2026). https://papers.nips.cc/paper_files/paper/2018/file/e5a764a045023b4913ecb81f65987f89-Paper.pdf

[54] Bartz-Beielstein, T., Doerr, C., Berg, D. V. D., Bossek, J., Chandrasekaran, S., Eftimov, T., ... & Weise, T. (2020). Benchmarking in optimization: Best practice and open issues. *arXiv preprint arXiv:2007.03488*.