REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE

MINISTERE DE L'ENSEIGNEMENT SUPERIEUR ET DE LA RECHERCHE SCIENTIFIQUE

Université de Mohamed El-Bachir El-Ibrahimi - Bordj Bou Arreridj

Faculté *des Sciences et de la technologie*

Département Electronique

Mémoire

Présenté pour obtenir

LE DIPLOME DE MASTER

FILIERE : Electronique

Spécialité: Electronique des Systèmes Embarqués

Par

- **MOKHTARI Abderrahime**
- **KHATTARA Seddik**

Intitulé

Etude d'un système de reconnaissance des chiffres manuscrits basé sur la l'apprentissage profond et le calcul d'ordre fractionnaire

Soutenu le: 25/06/2024

Devant le Jury composé de :

Nom & Prénom	*Grade*	*Qualité*	*Etablissement*
Dr. DIFFELLAH Nacira	*MCA*	*Président*	*Univ-BBA*
Dr. TALBI Mohamed Lamine	*MCA*	*Encadreur*	*Univ-BBA*
Dr. BENAMER Ahmed	*MCB*	*Examinateur*	*Univ-BBA*

Année Universitaire 2023/2024

Remerciements

Avant tout on remercie le Dieu tout puissant pour sa générosité de nous donner la santé, la volonté et la patience tout au long de la vie. Nous exprimons notre profonde gratitude au « Dr. Talbi Mohamed Lamine », notre encadreur, qui nous a suggéré ce sujet et qui nous a guidés et conseillés tout au long de sa réalisation. Monsieur Talbi a fait preuve d'une grande disponibilité et d'un grand soutien pour nous aider à progresser et à enrichir notre travail. Nous adressons nos sincères remerciements à nos enseignants ; sans oublier aussi leurs conseils permanents, leur orientation efficace et leur grande patience. On tient à exprimer notre profonde gratitude aux membres de jury. Nous remercions avec sincérité toutes les personnes qui ont contribué de près ou de loi à la réalisation de ce travail

Dédicaces

Avant de commencer, nous voulons remercier Allah pour tout ce que nous avons fait et pour tout ce que nous le ferons.

Nous dédions ce travail à nos chers parents qui n'ont cessé de nous encourager tout au long de nos études.

À mes frères et sœurs, mes cousins et cousines.

À mes amis et à tous ceux qui m'ont aidé à terminer ce travail.

Et finalement à tous les étudiants d'électronique.

Résumé

Ce mémoire explore la reconnaissance des chiffres manuscrits en combinant l'apprentissage profond et le calcul d'ordre fractionnaire. L'objectif principal est d'évaluer l'impact de l'intégration du calcul fractionnaire dans les règles de mise à jour des poids lors de l'apprentissage des réseaux de neurones. Diverses méthodes d'optimisation, architectures de réseaux et modifications sont étudiées. Les résultats expérimentaux démontrent la supériorité des optimiseurs fractionnaires, en particulier FAdam, par rapport aux optimiseurs classiques. Les réseaux de neurones convolutionnels avec trois couches cachées couplés à FAdam offrent les meilleures performances pour la reconnaissance des chiffres manuscrits de la base MNIST. Cette approche ouvre de nouvelles perspectives pour l'intégration du calcul fractionnaire dans l'apprentissage profond et l'intelligence artificielle.

Abstract

This thesis explores handwritten digit recognition by combining deep learning and fractional calculus. The main objective is to evaluate the impact of integrating fractional calculus into the weight update rules during the training of neural networks. Various optimization methods, network architectures, and modifications are studied. The experimental results demonstrate the superiority of fractional optimizers, particularly FAdam, over classical optimizers. Convolutional neural networks with three hidden layers coupled with FAdam offer the best performance for handwritten digit recognition on the MNIST dataset. This approach opens new avenues for integrating fractional calculus into deep learning and artificial intelligence.

ملخص

يستكشف هذا البحث التعرف على الأرقام المكتوبة بخط اليد من خلال الجمع بين التعلم العميق وحساب الكسور. الهدف الرئيسي هو تقييم تأثير دمج حساب الكسور في قواعد تحديث الأوزان أثناء تدريب الشبكات العصبية، تم دراسة العديد من طرق التحسين والهياكل الشبكية والتعديلات المختلفة، تُظهر النتائج التجريبية تفوق المحسنات الكسورية، وخاصةFAdam ، على المحسنات التقليدية. توفر الشبكات العصبية التفافية ذات الطبقات الثلاث المخفية المرتبطة ب FAdam، أفضل أداء للتعرف على الأرقام المكتوبة بخط اليد في قاعدة بيانات .MNIST تفتح هذه المقاربة آفاقًا جديدة لدمج حساب الكسور في التعلم العميق والذكاء الاصطناعي.

Table des matières

Liste des figures :

Liste des Tableaux :

Liste des abréviations (ordre alphabetique)

CNN Convolutional Neural Network

FSGD Fractional Stochastic Gradient Descent

FAdam Fractional Adaptive Moment Estimation

FAdadelta Fractional Adaptatif delta

FRMSProp Fractional Root Mean Square (Racine carrée de la moyenne)

FAdagrad Fractional Adaptatif grad

MLP Multi-Layer Perceptron

MNIST Modified National Institute of Standards and Technolo

Introduction générale

La reconnaissance des chiffres manuscrits représente un domaine clé de l'intelligence artificielle, ayant des implications significatives dans divers secteurs tels que la reconnaissance optique de caractères, la numérisation de documents et la vérification de signatures. Avec l'avènement de l'apprentissage profond et du calcul d'ordre fractionnaire, de nouvelles opportunités se sont ouvertes pour améliorer la précision et la robustesse des systèmes de reconnaissance.

L'apprentissage profond, une sous-branche de l'intelligence artificielle inspirée par le fonctionnement du cerveau humain, a révolutionné la reconnaissance de motifs complexes dans les données. En utilisant des architectures de réseaux de neurones profonds, les systèmes d'apprentissage profond sont capables d'apprendre des représentations hiérarchiques de l'information, ce qui leur permet de reconnaître des motifs abstraits et d'effectuer des tâches complexes telles que la reconnaissance de chiffres manuscrits avec une précision accrue.

D'autre part, le calcul d'ordre fractionnaire élargit le cadre traditionnel des opérations mathématiques en incluant des concepts de dérivées et d'intégrales fractionnaires. Cette approche permet de capturer des caractéristiques plus fines et plus subtiles dans les données, offrant ainsi une représentation plus riche. En intégrant le calcul d'ordre fractionnaire dans les modèles d'apprentissage profond, il devient possible de mieux exploiter les informations disponibles, ce qui peut conduire à des améliorations significatives en termes de précision et de généralisation.

Ce travail vise à classifier les chiffres manuscrits en utilisant des réseaux de neurones artificiels. L'objectif principal est d'évaluer l'impact de l'intégration du calcul d'ordre fractionnaire dans les règles de mise à jour des poids lors de l'apprentissage. Diverses règles de mise à jour des poids seront explorées et testées pour déterminer leur efficacité. Nous comparerons plusieurs méthodes d’optimisation utilisées pour l'adaptation des poids de l’algorithme d’apprentissage, en explorant des modifications et des pistes de recherche non considérées dans les études précédentes. De plus, nous évaluerons l'impact de l'architecture et du type de réseau de neurones sur les résultats de classification.

Ce mémoire est structuré en quatre chapitres distincts. Le premier chapitre introduit les notions fondamentales de la classification des chiffres manuscrits, établissant les bases théoriques nécessaires. Le second chapitre explore en profondeur les réseaux de neurones, une approche clé pour la reconnaissance de formes. Le troisième chapitre se concentre sur le

calcul d'ordre fractionnaire, un domaine mathématique essentiel pour modéliser des phénomènes complexes. Enfin, le quatrième chapitre présente les résultats expérimentaux et l'application pratique de notre travail.

Chapitre I

La reconnaissance des chiffres manuscrites

1. Introduction

Le présent chapitre se penche sur la reconnaissance des chiffres manuscrits, une branche cruciale de l'apprentissage automatique et de la vision par ordinateur. Cette discipline revêt une importance significative dans divers secteurs tels que le tri postal, le traitement des chèques et la gestion des documents numérisés. Nous explorerons en détail les différentes étapes impliquées dans ce processus, mettant en lumière les méthodes et les défis rencontrés lors de la reconnaissance des chiffres écrits à la main.

2. La reconnaissance des chiffres manuscrits

La reconnaissance des chiffres manuscrits, également appelée "Hand Written digit recognition" en anglais, est un domaine de la reconnaissance de formes et de l'apprentissage automatique qui se concentre sur le développement d'algorithmes pour reconnaître et interpréter les chiffres écrits à la main. Cette technologie a de nombreuses applications, notamment dans l'automatisation postale, le traitement des chèques bancaires et la gestion des documents numérisés [1].

Ce système est fondé sur les techniques d'apprentissage automatique, se compose de cinq phases cruciales :

2.1 Prétraitement

Les étapes de prétraitement consistent à sélectionner l'information nécessaire dans l'espace de représentation pour l'application visée. Cette sélection inclut souvent l'élimination du bruit causé par les conditions d'acquisition. Les techniques développées à cet effet veillent à ne pas altérer les propriétés essentielles des formes, car toute modification pourrait entraîner de graves erreurs d'analyse et, par conséquent, de reconnaissance [2]. Ainsi, le rôle du prétraitement est de préparer les données reçues du capteur pour la phase suivante d'analyse, dédiée à l'extraction des paramètres [3].

Parmi les opérations de prétraitement, on peut citer : la binarisation, l'élimination du bruit, la squelettisation et la normalisation.

2.1.1 La binarisation

Dans un système de reconnaissance d’écriture, la numérisation des images est la première étape de traitement. Elle permet de convertir une image en niveaux de gris en une image binaire composée de deux valeurs, 0 et 1, plus simples à traiter. En général, un seuil de binarisation approprié est utilisé pour distinguer les contrastes forts et faibles dans l’image.

Selon la méthode de calcul du seuil de binarisation, on distingue deux types de binarisation : le seuillage global et le seuillage adaptatif [4].

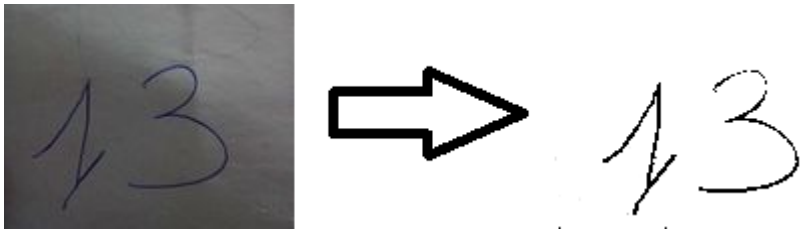

Figure I.1: Exemple de binarisation

2.1.2 Elimination de bruit

Le bruit est un problème très important et difficile à résoudre, notamment dans le cas des documents ayant un fond complexe, comme le papier de couleur. Une méthode simple pour traiter ce problème consiste à effectuer la soustraction entre l'image d'entrée et l'image d'un papier de couleur sans écriture [5].

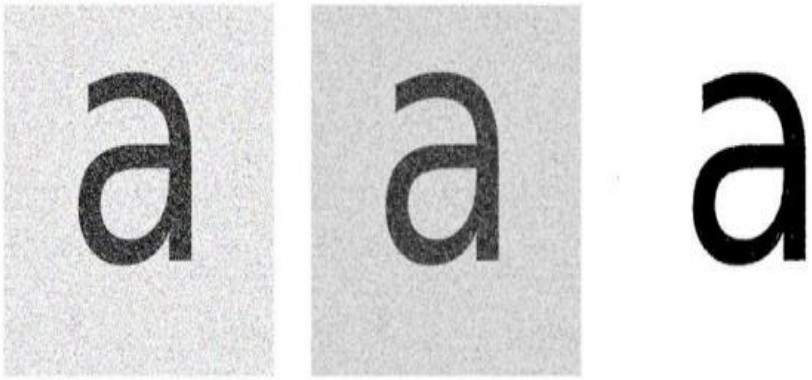

Figure I.2: Exemple d'élimination de bruit d'une image [11]

2.1.3 La squelettisation

Le processus de squelettisation permet de réduire et de compacter la taille de l'image, et trouver un axe médian, définit comme l'ensemble de pixels S qui possèdent une égalité de distance par rapport aux pixels de frontière qui les entourent. La sortie de ce processus est un squelette d'un chiffre manuscrit.[5]

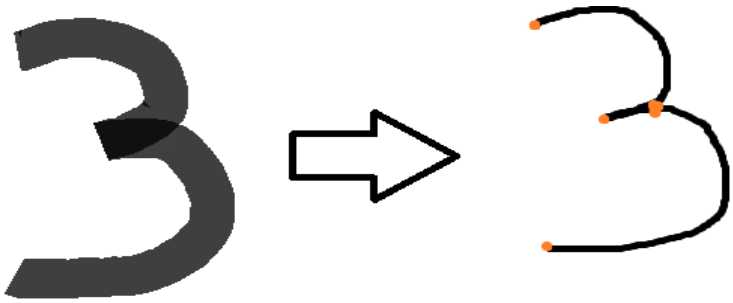

Figure I.3: Exemple de La squelettisation

2.1.4 La normalisation

Le principe de la normalisation consiste à uniformiser localement les différentes parties du mot afin d'augmenter la ressemblance entre les images. Cette opération introduit généralement de légères déformations sur les images. Cependant, certains traits caractéristiques, tels que la hampe dans les caractères (Ø, Ù, á, Ç, par exemple), peuvent être éliminés à la suite de la normalisation, ce qui peut entraîner des confusions entre certains caractères [6].

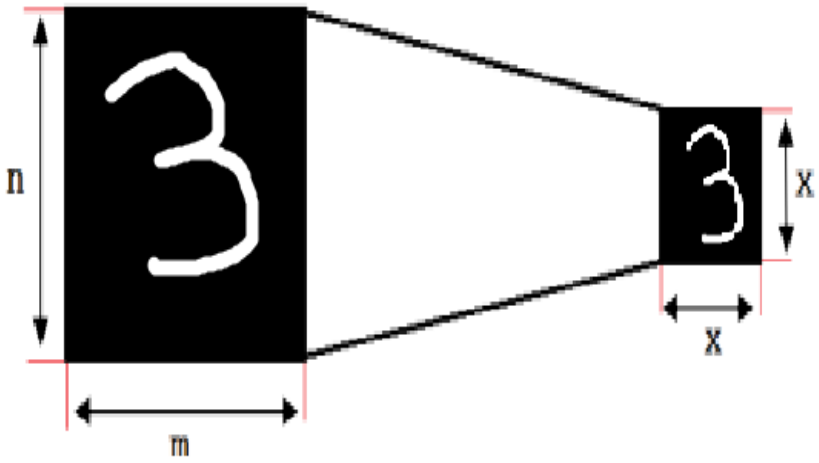

Figure I.4: Exemple de La normalisation

2.2 Segmentation

La phase de segmentation consiste à diviser une image contenant plusieurs chiffres manuscrits en régions distinctes, chacune correspondant à un chiffre individuel. Cette étape est cruciale pour isoler chaque chiffre et permettre leur analyse individuelle [7].

Il existe deux grandes catégories de segmentation :

- La Segmentation explicite.
- La Segmentation implicite.

2.2.1 Segmentation explicite

Dans cette approche, la séparation des chiffres consiste à diviser la chaîne des chiffres en plusieurs images aussi parfaitement que possible. Les chemins de segmentation sont généralement obtenus par des points caractéristiques, tels que [8]:

- Des minima locaux du contour supérieur.
- Des espaces entre les caractères ou bien les sous-mots.
- Des points d'intersection les plus probables par une analyse des composantes dans le mot.

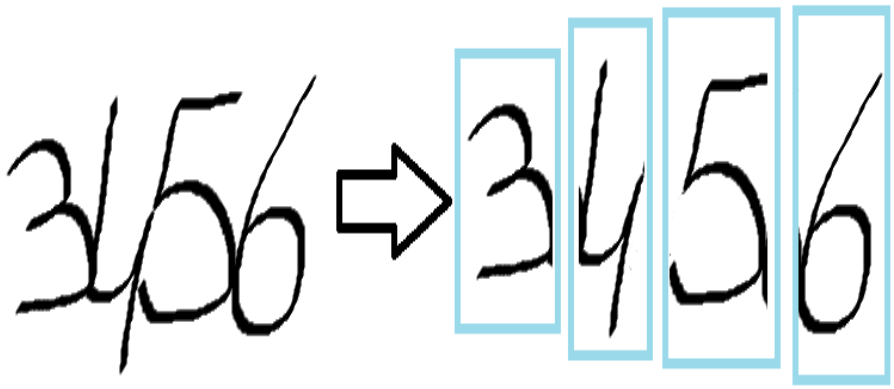

Figure I.5: Exemple de segmenation explicite

2.2.2 Segmentation implicite

La segmentation implicite fait référence à une approche où les limites entre les chiffres ne sont pas explicitement définies par des règles ou des algorithmes prédéfinis. Au lieu de cela, des techniques telles que la segmentation basée sur la connectivité des pixels ou l'utilisation de méthodes de regroupement sont utilisées pour diviser l'image en régions contenant des chiffres [9].

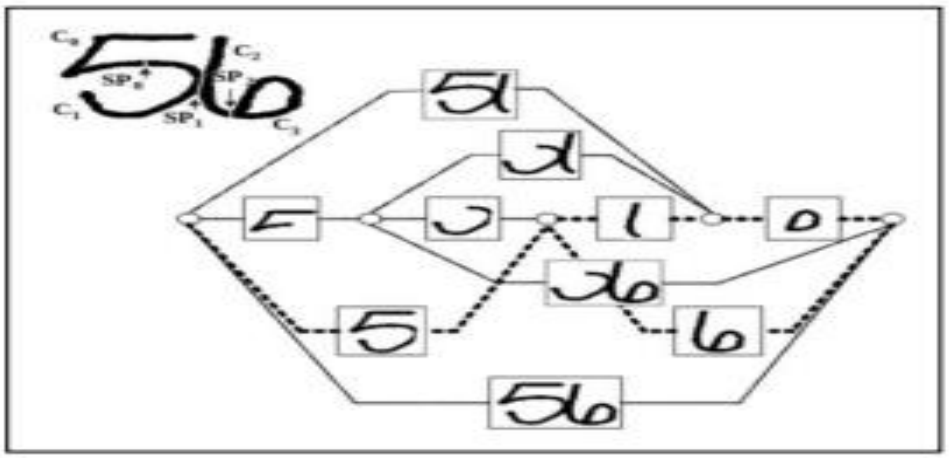

Figure I.6 : Exemple de segmenation implicite [22]

2.3 Extraction des caractéristiques

L'objectif de l'extraction des caractéristiques dans le domaine de la reconnaissance consiste à exprimer les primitives sous une forme numérique ou symbolique appelée codage. Selon le cas, les valeurs de ces primitives peuvent être réelles, entières ou binaires [2].

La difficulté ici est de trouver de bonnes caractéristiques. De « bonnes » caractéristiques permettent aux classifieurs de reconnaître facilement les différentes classes d'objets ; on dit alors qu'elles sont discriminantes. Elles doivent aussi être invariantes à certaines

transformations : « le chiffre 1 appartient à la même classe quelle que soit sa taille et sa structure » [10].

2.3.1 Caractéristiques structurelles

Les caractéristiques structurelles de l'image peuvent inclure la forme des caractères, par exemple le contour, les points finaux, les points de branchement, les boucles, les directions caractéristiques, et autres [11]. Une description structurale d'une texture implique la recherche des motifs élémentaires, leur description, puis la détermination des règles conditionnant leur position [12].

2.3.2 Caractéristiques globale

Les caractéristiques globales cherchent à représenter au mieux la forme générale d'un objet et sont donc calculées sur des images relativement grandes (par exemple, les histogrammes de couleur, les moments statistiques, la cohérence spatiale comme les histogrammes de connexité, etc., pour la texture et la forme). Les caractéristiques locales sont calculées lors d'un parcours des pixels de l'image avec un pas d'analyse qui dépend de la modélisation, du type de caractéristique et de la taille de l'image [13].

2.3.3 Caractéristiques morphologique

"Caractéristiques morphologiques" désignent les propriétés de forme et les structures des objets présents dans une image. Ces caractéristiques sont utilisées pour décrire les aspects morphologiques tels que la taille, la forme et la disposition spatiale des objets dans une image [14].

2.3.4 Caractéristiques texturale

Les caractéristiques texturales sont des propriétés locales des images qui décrivent les motifs ou la texture présente dans une région donnée. Elles sont utilisées pour capturer les variations de densité, de rugosité ou de régularité des pixels, fournissant ainsi des informations sur la structure interne des objets dans l'image [21].

2.4 Classification

La classification dans un système de reconnaissance consiste à attribuer une donnée à une classe prédéfinie. Cette étape regroupe deux tâches : l'apprentissage et la décision. Elle vise à associer une forme de donnée à un modèle de référence. Le résultat de l'apprentissage peut être la réorganisation ou le renforcement des modèles existants en prenant en compte l'ajout d'une nouvelle forme, soit par la création d'un modèle, soit par l'apprentissage. Le résultat de

la décision est un "avis" sur l'appartenance ou non de la forme aux modèles d'apprentissage [15].

2.4.1 L'apprentissage

Cette étape permet de construire un dictionnaire de prototypes. Il s'agit de regrouper en classes plusieurs prototypes dont les caractéristiques se rapprochent.

2.4.1.1 L'apprentissage supervisé

Cette classification est guidée par un superviseur appelé professeur. Sa réalisation se fait lors de la phase de reconnaissance par l'introduction d'un grand nombre d'échantillons de référence. Le professeur indique le nom des échantillons. Le choix des primitives de référence est effectué manuellement en fonction du système. Le nombre d'échantillons peut varier de quelques unités à quelques dizaines, voire quelques centaines par caractère [16].

2.4.1.2 L'apprentissage non supervisée

L'apprentissage non supervisé, ou sans professeur, consiste à doter le système d'un mécanisme automatique qui s'appuie sur des règles précises de regroupement pour trouver les classes de référence avec une assistance minimale. Dans ce cas, les échantillons sont introduits en grand nombre par l'utilisateur sans indiquer leur classe [17].

2.4.2 L'approche de reconnaissance

Les données d'apprentissage sont celles utilisées pour construire un classificateur capable de reconnaître des formes inconnues afin de caractériser les classes [18].

2.4.2.1 Approche statistique

Elle repose sur l'étude statistique des mesures effectuées sur les formes à reconnaître, en examinant leur répartition dans un espace métrique et en caractérisant statistiquement les classes. Cela aide à prendre une décision de reconnaissance basée sur la "plus forte probabilité d'appartenance à une classe" [15]. Cependant, elle nécessite un nombre élevé d'exemples pour effectuer un apprentissage correct des lois de probabilité des différentes classes.

2.4.2.2 Approche structurelle

Les méthodes structurelles s'appuient sur la structure physique des caractères. Elles visent à identifier des éléments simples ou primitifs tels que les graphèmes ou même les pixels de l'image, et à décrire leurs relations [19].

2.4.2.3 Approche stochastique

Cette approche utilise un modèle de reconnaissance qui prend en compte la grande variabilité de la forme. Plutôt que d'utiliser la distance commune dans les techniques de

comparaison dynamique, elle utilise des probabilités calculées de manière plus précise par l'apprentissage. La forme est considérée comme un signal continu observable dans le temps à différents endroits, constituant des états "d'observations". Le modèle représente ces états à l'aide de calculs de probabilités de transitions d'états et de probabilités d'observation par état. La comparaison consiste à trouver, dans ce graphe d'états, le chemin de probabilité forte correspondant à une suite d'éléments observés dans la chaîne d'entrée [15].

2.4.2.4 Approche hybride

Pour améliorer les performances de reconnaissance, la tendance actuelle consiste à mélanger différents types de caractéristiques et à combiner plusieurs classifieurs en couches au sein d'un seul système. Cette approche vise à surmonter les faiblesses de chaque méthode et à obtenir des résultats plus précis que ceux qui auraient été obtenus en appliquant chaque méthode séparément. Par exemple, des approches fusionnées ont été combinées pour former un système intégré [20].

3. Conclusion

Ce chapitre a exploré la reconnaissance de l'écriture manuscrite, en mettant particulièrement l'accent sur la reconnaissance des chiffres. Nous avons examiné en détail les étapes cruciales telles que la binarisation, la segmentation et l'extraction des caractéristiques, nécessaires pour préparer les images avant la classification. De plus, nous avons étudié différentes approches de classification, allant de l'apprentissage supervisé à l'approche hybride, afin d'identifier et de classifier précisément les chiffres dans les images. Cette analyse approfondie fournit une base solide pour comprendre le processus complexe de reconnaissance de l'écriture manuscrite et ses applications dans divers domaines de l'informatique et de l'intelligence artificielle.

Dans le prochain chapitre, nous aborderons les principes et les fondements du Deep Learning, également connu sous le nom d'apprentissage profond.

Chapitre II

L'apprentissage profond «Deep Learning»

1. Introduction

Le Deep Learning, également connu sous le nom d'apprentissage profond, est une discipline de l'intelligence artificielle qui a attiré beaucoup d'attention ces dernières années en raison de ses performances remarquables dans divers domaines. Cette technologie novatrice a révolutionné notre approche des problèmes complexes liés au traitement des données, à la reconnaissance de motifs et à la prise de décision.

Au cœur du Deep Learning se trouvent les réseaux de neurones artificiels, des modèles informatiques inspirés du fonctionnement du cerveau humain. Ces réseaux sont capables d'apprendre à partir de données brutes en identifiant automatiquement des caractéristiques et en ajustant leurs paramètres internes pour accomplir des tâches spécifiques, telles que la reconnaissance d'images, la traduction automatique, la génération de texte, et bien d'autres encore.

Dans ce chapitre, nous explorerons les bases fascinantes du Deep Learning, en examinant ses principes fondamentaux, ses applications révolutionnaires et les défis uniques qu'il pose.

2. Définition

L'apprentissage profond est un sous-ensemble de l'apprentissage automatique, lui-même un sous-ensemble de l'intelligence artificielle (IA). Cela implique de former des réseaux de neurones artificiels sur de grandes quantités de données pour reconnaître des modèles et faire des prédictions. Contrairement aux algorithmes d'apprentissage automatique traditionnels, dont les performances plafonnent souvent à mesure que le volume de données augmente, les modèles d'apprentissage profond continuent de s'améliorer avec davantage de données et de calculs. Ces modèles, en particulier les réseaux de neurones profonds, sont constitués de plusieurs couches de nœuds interconnectés (neurones), chaque couche transformant les données d'entrée en une représentation plus abstraite et composite [28].

3. Les réseaux de neurones artificiels

Les réseaux de neurones artificiels « RNA » sont des réseaux fortement connectés de processeurs élémentaires fonctionnant en parallèle. Chaque processeur élémentaire calcule une sortie unique sur la base des informations qu'il reçoit. Toute structure hiérarchique de réseaux est évidemment un réseau [23].

3.1 Le perceptron

Le perceptron « un seul neurone » est un modèle de réseau de neurones artificiels avec algorithme d'apprentissage [24], voire la « Figure 1 ».

Les composants de base d'un perceptron sont :

- Couche d'entrée : la couche d'entrée se compose d'un ou plusieurs neurones d'entrée, qui reçoivent des signaux.
- Poids : Chaque neurone d'entrée est associé à un poids, qui représente la force de la connexion entre le neurone d'entrée et le neurone de sortie.
- Biais : un biais est ajouté à la couche d'entrée pour fournir au perceptron une flexibilité supplémentaire dans la modélisation de modèles complexes dans les données d'entrée.
- Fonction d'activation : La fonction d'activation détermine la sortie du perceptron en fonction de la somme pondérée des entrées et du terme de biais. Les fonctions d'activation courantes utilisées dans les perceptrons incluent la fonction sigmoïde et la fonction ReLU.
- Sortie : La sortie du perceptron est une valeur binaire unique, 0 ou 1, qui indique la classe ou la catégorie.

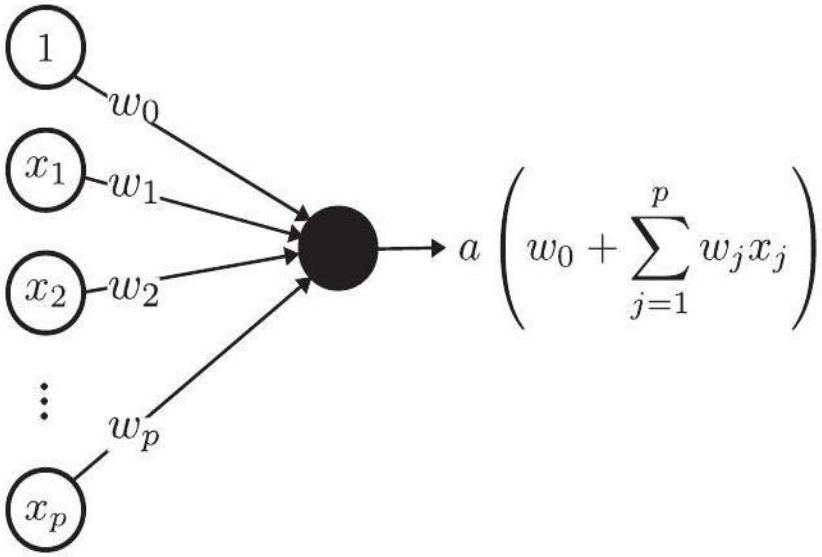

Figure II.1 : 1 - Architecture d'un perceptron.

3.2 Perceptrons multicouche

Un réseau de neurones multicouche est un type de réseau de neurones artificiels comprenant plusieurs couches de neurones, généralement organisées en une couche d'entrée, une ou plusieurs couches cachées, et une couche de sortie. Chaque neurone dans une couche est connecté à tous les neurones de la couche précédente et de la couche suivante, mais il n'y a pas de connexions entre les neurones de la même couche. Ce modèle permet au réseau d'apprendre des représentations de données complexes en introduisant des non-linéarités à travers des fonctions d'activation dans les couches cachées. Les réseaux de neurones multicouches sont largement utilisés dans diverses tâches d'apprentissage automatique, y compris la classification, la régression et la génération de contenu [28].

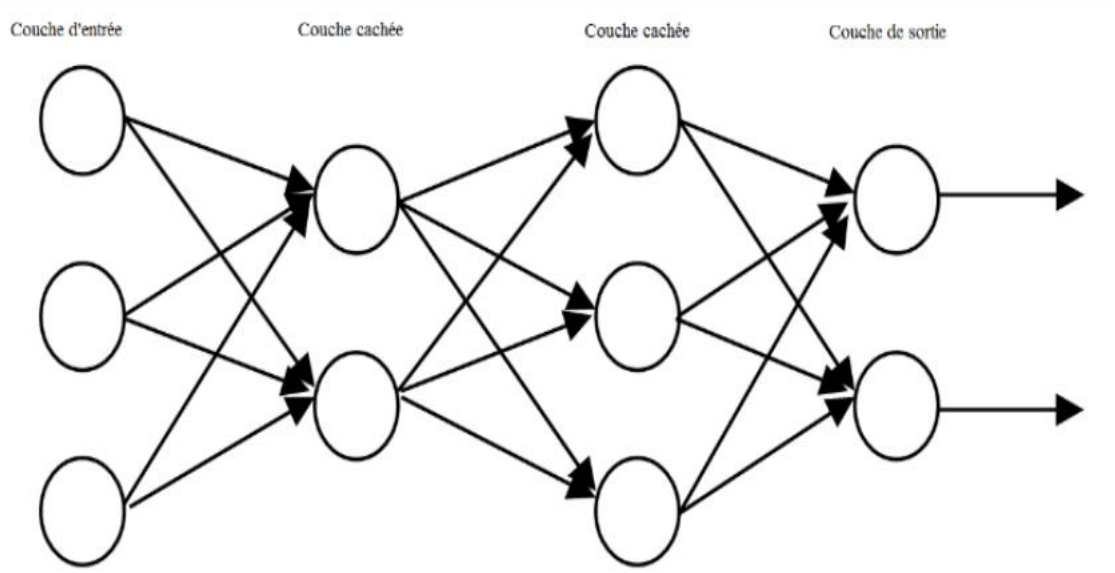

Figure II.2 : Le perceptron multicouche

4. Fonction d'activations

Une fonction d'activation est une opération mathématique appliquée à la sortie de chaque neurone. Elle introduit de la non-linéarité, permettant aux réseaux neuronaux de modéliser des motifs complexes dans les données.

À titre d'exemple, voici quelques fonctions couramment utilisées dans le domaine des réseaux de neurones :

- La fonction de Signe doux (1) qui renvoie 1 si le nombre est strictement positif, 0 si le nombre est nul, et -1 si le nombre est strictement négatif [25] :

$$\forall x \in R, \operatorname{sgn}(x) = \begin{cases} -1 \text{ si } x < 0 \\ 0 \text{ si } x = 0 \\ 1 \text{ si } x > 0 \end{cases} \quad (1)$$

- La fonction sigmoïde (2) est utilisée pour le calcul de la valeur de sortie (x) d'un neurone artificiel. Celle-ci est de forme [26] :

$$f(x) = \frac{1}{1 + e^{-x}} \quad (2)$$

- La fonction linéaire (3) c'est une fonction simple de la forme [27] :

$$\begin{cases} f(x) = ax \\ \text{ou} \\ f(x) = x \end{cases} \quad (3)$$

- Fonction ReLu (4) est une fonction appréciée pour sa simplicité et son efficacité, car elle permet d'accélérer la convergence de l'apprentissage en évitant le problème de disparition du gradient. Celle-ci est de forme [28] :

$$f(x) = max(0, x) \tag{4}$$

- Fonction softmax (5) Mathématiquement, la fonction softmax prend un vecteur de scores arbitraires à valeur réelle **z**=*(z1,z2,.....,zk)* et génère un vecteur de probabilités **p**=*(p1,p2,...,pk)* tel que[28] :

$$pi = \frac{e^{zi}}{\sum_{j=1}^{k} e^{zj}} \tag{5}$$

5. Rétropropagation du gradient «Back Propagation »

La rétropropagation du gradient est une technique essentielle dans l'apprentissage supervisé des réseaux de neurones. Elle permet de calculer les gradients des poids du réseau par rapport à une fonction de perte. Cette technique permet ensuite de mettre à jour les poids du réseau de manière à minimiser la perte lors de l'apprentissage [29].

6. Réseau de neurones convolutifs « CNNs »

Les réseaux de neurones convolutionnels (CNN), également connus sous le nom de ConvNets, sont une architecture de réseau neuronal profond largement utilisée pour la reconnaissance d'images. Ils se caractérisent par l'utilisation de couches de convolution qui appliquent des filtres pour extraire des caractéristiques importantes des images en entrée. Les CNN sont particulièrement efficaces pour capturer les motifs spatiaux locaux dans les images, ce qui les rend adaptés à une variété de tâches telles que la classification d'images, la détection d'objets et la segmentation sémantique [30].

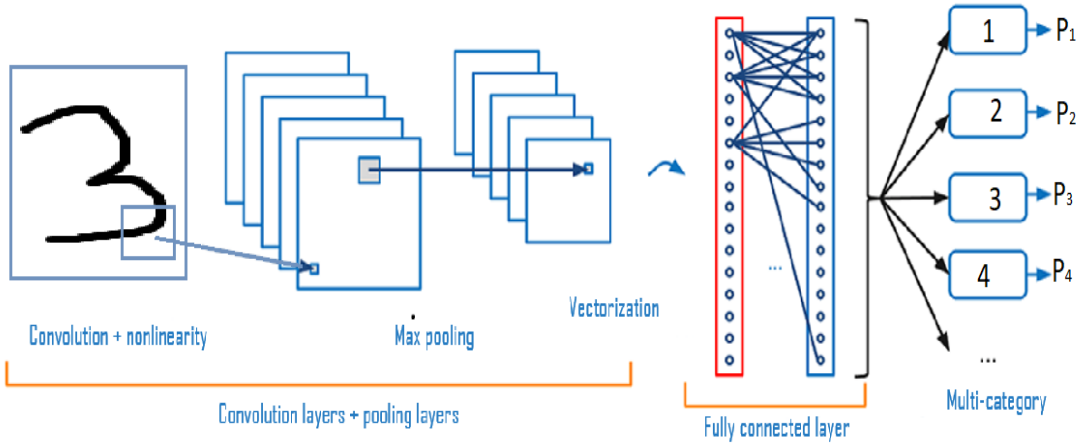

Figure II.3 : Schéma fonctionnel typique d'un CNNs

Les quatre catégories de couches les plus fréquemment rencontrées sont les suivantes :

6.1 Couche de convolution

La convolution est la première couche qui extrait des entités d'une image d'entrée. Elle préserve la relation entre les pixels en apprenant les caractéristiques de l'image à l'aide de petits carrés de données d'entrée. Il s'agit d'une opération mathématique qui prend deux entrées telles qu'une matrice d'image et un filtre ou un noyau [34].

6.2 Couches de 'Pooling' (regroupement)

Une couche de regroupement, également appelée couche de pooling, est une composante courante des réseaux de neurones convolutionnels (CNN). Son rôle principal est de réduire la dimension spatiale des représentations en entrée, tout en préservant les caractéristiques importantes. Cela se fait en regroupant les activations des neurones voisins dans une région donnée en une seule valeur de sortie, généralement en prenant le maximum (max pooling) ou la moyenne (average pooling).

Les couches de regroupement aident à rendre la représentation des données plus invariante aux petites translations spatiales de l'entrée, ce qui rend le réseau plus robuste et moins sensible aux variations mineures dans les données d'entrée [32].

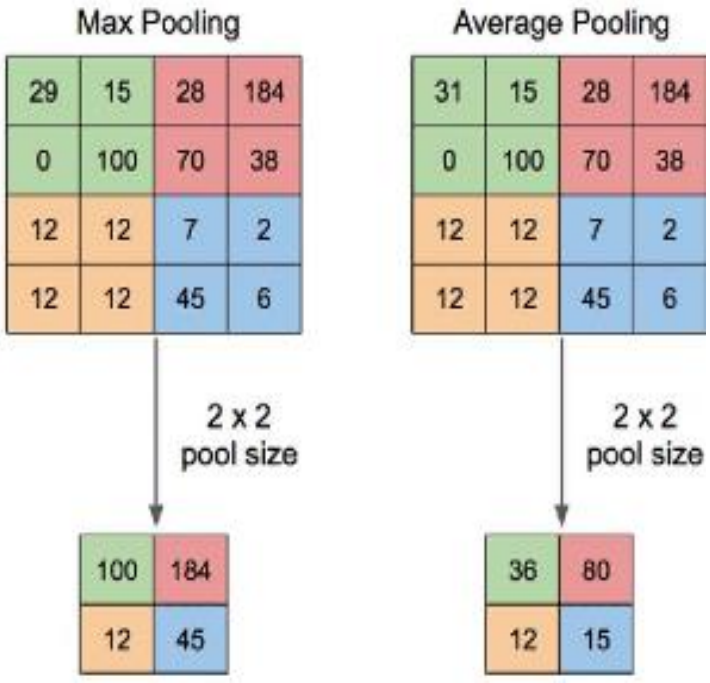

Figure II.4 :« max pooling » Et « average pooling »[35]

6.3 Couches non linéaires

Les couches non linéaires dans les réseaux de neurones sont des composantes essentielles qui introduisent de la non-linéarité dans le flux d'informations. Elles sont cruciales pour permettre au réseau de modéliser des relations complexes et des motifs non linéaires dans les

données. Une couche non linéaire applique une fonction d'activation à la sortie de chaque neurone, transformant ainsi les données en une forme non linéaire. Cela permet au réseau de capturer des structures et des caractéristiques plus complexes des données [31].

6.4 Couches entièrement connectées

C'est la structure d'interconnexion la plus générale. Chaque neurone est connecté à tous les neurones du réseau (et à lui-même) [33].

7. Conclusion

Ce chapitre a exploré les bases de l'apprentissage profond, également connu sous le nom de Deep Learning. Nous avons commencé par une introduction générale sur ce sujet, suivie d'une définition détaillée de ce concept en émergence. Ensuite, nous avons examiné les réseaux de neurones artificiels, en mettant en évidence leur structure fondamentale, incluant le perceptron et les perceptrons multicouches. Nous avons également souligné l'importance des fonctions d'activation dans le processus d'apprentissage des réseaux neuronaux.

Une part de ce chapitre a été consacrée à la rétropropagation du gradient, une technique cruciale dans l'entraînement des réseaux de neurones. Par la suite, nous avons approfondi notre compréhension des réseaux de neurones convolutifs (CNNs), en mettant en lumière leur architecture en couches, notamment les couches de convolution, de pooling, les couches non linéaires et les couches entièrement connectées.

Chapitre III

Calcul Fractionnaire

1. Introduction

Dans le domaine en constante évolution du Deep Learning, les calculs fractionnaires jouent un rôle essentiel dans la manipulation précise des données et des paramètres des réseaux neuronaux. Bien que les réseaux de neurones convolutifs (CNNs) et les réseaux récurrents aient révolutionné le traitement des données complexes telles que les images et les séquences, l'utilisation des calculs fractionnaires offre une flexibilité supplémentaire pour affiner la précision et l'efficacité de ces modèles.

Dans ce chapitre consacré au calcul fractionnaire, nous explorons un domaine mathématique émergent qui propose de nouvelles perspectives pour la résolution de problèmes complexes.

2. Définition

Le calcul fractionnaire est une branche des mathématiques qui généralise les opérations classiques d'intégration et de dérivation à des ordres non entiers. Il permet de manipuler des fonctions et des opérateurs dont les degrés sont des nombres fractionnaires. Cette extension du calcul classique est largement utilisée dans divers domaines scientifiques, notamment en physique, en ingénierie, en économie et en biologie, pour modéliser des phénomènes complexes et non linéaires [36].

3. Fonction Gamma d'Euler

La fonction gamma d'Euler, souvent simplement appelée la fonction gamma, est une fonction mathématique notée $\Gamma(z)$ définie pour tout nombre complexe z avec une partie réelle positive selon l'intégrale suivante [37]:

$$\Gamma(z) = \int_0^\infty t^{z-1}e^{-t}dt \qquad z > 0. \tag{1}$$

Cette fonction généralise la notion de factorielle pour des valeurs non entières.

4. Fonction Beta

La fonction bêta, notée $B(x, y)$, est une fonction définie pour deux nombres réels positifs x et y. Elle est définie comme suit [37]:

$$B(x,y) = \int_0^1 t^{x-1}(1-t)^{y-1}dt \qquad \text{x>0 , y>0.} \tag{2}$$

La fonction bêta est souvent utilisée en conjonction avec la fonction gamma d'Euler est sont reliées par la relation :

$$B(x,y) = \frac{\Gamma(\text{x})\Gamma(\text{y})}{\Gamma(\text{x}+\text{y})} \tag{3}$$

5. Intégration fractionnaire

L'intégration fractionnaire est une généralisation de l'intégration ordinaire qui permet d'étendre les opérations d'intégration à des ordres non entiers. Plutôt que d'intégrer une fonction sur un intervalle défini, l'intégration fractionnaire consiste à intégrer une fonction sur un ordre fractionnaire [36].

5.1 Intégrale fractionnaire de Riemann –Liouville

L'intégrale fractionnaire de Riemann-Liouville est une généralisation de l'intégrale de Riemann qui permet d'étendre les opérations d'intégration à des ordres non entiers. Cette intégrale est définie pour une fonction comme suit :

$$\frac{1}{\Gamma(\alpha)}\int_{a}^{x}(x-t)^{\alpha-1}f(t)dt \tag{4}$$

où α est un nombre réel positif, α est la borne inférieure de l'intégration, x est la borne supérieure de l'intégration et $\Gamma(\alpha)$ est la fonction gamma d'Euler.

Cette intégrale généralisée permet de modéliser des phénomènes complexes et non linéaires qui ne peuvent pas être représentés efficacement par des méthodes traditionnelles d'intégration [36].

6. Dérivation fractionnaire

La dérivation fractionnaire étend le concept traditionnel de dérivation à des ordres non entiers, permettant ainsi d'analyser les variations d'une fonction sur des intervalles fractionnaires. Contrairement à la dérivation classique, qui se limite aux ordres entiers, la dérivation fractionnaire capture les variations continues d'une fonction, offrant une perspective plus précise sur son comportement dynamique.

Cette extension du concept de dérivation est largement utilisée dans divers domaines scientifiques et techniques, notamment en physique, en ingénierie, en finance et en biologie. Elle permet de modéliser des phénomènes complexes tels que la diffusion anormale, les processus de relaxation et les systèmes dynamiques non linéaires [38].

6.1 Approche de Riemann –Liouville

L'approche de Riemann-Liouville pour la dérivation fractionnaire est l'une des méthodes les plus couramment utilisées pour étendre la dérivation traditionnelle à des ordres non entiers. Elle est définie par l'intégrale de Riemann-Liouville de la fonction $f(t)$ comme suit :

$$R_{L\,a+}^{\alpha}f(t)=\frac{1}{\Gamma(n-\alpha)}\frac{d^{n}}{dt^{n}}\int_{a}^{t}(t-\tau)^{n-\alpha-1}f(\tau)d\tau \tag{5}$$

où α est un nombre réel positif, n est l'entier le plus proche de α, a est la borne inférieure de l'intégration, t est la borne supérieure de l'intégration et Γ est la fonction gamma d'Euler [36].

6.2 Approche de Caputo

L'approche de sens de Caputo pour la dérivation fractionnaire est une méthode alternative à l'approche de Riemann-Liouville. Elle est définie par la dérivée fractionnaire de Caputo de la fonction f(t) comme suit :

$$C_{D_a^{\alpha}+} f(t) = \frac{1}{\Gamma(n-\alpha)} \int_a^t (t-\tau)^{n-\alpha-1} f^n(\tau) d\tau \tag{6}$$

où α est un nombre réel positif, n est l'entier le plus proche de α, a est la borne inférieure de l'intégration et Γ est la fonction gamma d'Euler [36].

7. Exponentielle de Mittag-Leffler

La fonction Mittag-Leffler joue un rôle très important dans la théorie des équations différentielles d'ordre non entier. Elle est également largement utilisée dans la recherche des solutions des équations différentielles d'ordre fractionnaire. Cette fonction a été introduite par G. M. Mittag-Leffler [39].

Pour $z \in C$, la fonction de Mittag-Leffler Eα(z) est définie par [40]:

$$E_{\alpha}(z) = \sum_{k=0}^{\infty} \frac{z^k}{\Gamma(\alpha k+1)} \qquad \alpha > 0 \tag{7}$$

La fonction de Mittag-Leffler à deux paramètres joue également un rôle très important dans la théorie du calcul fractionnaire. Cette dernière a été introduite par Wiman [41] et elle est définie par le développement en série suivant :

$$E_{\alpha,\beta}(z) = \sum_{k=0}^{\infty} \frac{z^k}{\Gamma(\alpha k+\beta)} \qquad \alpha,\beta > 0 \tag{8}$$

8. Optimisation

Les réseaux neuronaux sont l'une des méthodes les plus appréciées et les plus performantes pour représenter les données. Les techniques de descente de gradient sont couramment employées pour optimiser ces réseaux. Le principe fondamental de la descente de gradient consiste à ajuster les paramètres du modèle dans la direction opposée à celle du gradient de la fonction objective, afin de minimiser cette fonction.

En se basant sur la règle de mise à jour des optimisateurs de descente de gradient, les paramètres libres sont mis à jour dans la direction opposée au gradient g_t sur la surface d'erreur d'approximation. Le taux d'apprentissage η module cette rétroaction pour avancer vers un minimum [42].

Les paramètres sont mis à jour dans une version batch de la descente de gradient (GD) en prenant en compte tous les échantillons d'entraînement. Cependant, cela est difficile à utiliser pour les grands ensembles de données. La formule de mise à jour pour la descente de gradient batch est :

$$\Delta\theta_t = -\eta g_t \tag{9}$$

Dans le cas contraire, une version de descente de gradient stochastique (SGD) actualise les paramètres pour chaque i-ième échantillon d'entraînement. La méthode de mise à jour de SGD est la suivante :

$$\Delta\theta_{t,i} = -\eta g_{t,i} \tag{10}$$

Bien qu'il puisse introduire des fluctuations, la descente de gradient stochastique (SGD) peut être utile pour explorer l'espace d'optimisation. Cependant, cela peut également introduire une variance inutile dans les mises à jour des paramètres, rendant le taux d'apprentissage un facteur critique. Pour pallier ces problèmes, une version mixte appelée descente de gradient par mini-batch propose de diviser l'ensemble de données en sous-ensembles pour atténuer cette variation [43].

Adagrad et al [44] propose une version évoluée de la descente de gradient qui adapte le taux d'apprentissage en fonction de la mémoire des gradients. Adagrad calcule une matrice diagonale historique $G_{t,ii}$, accumulant la somme des carrés des gradients pour modifier l'ajustement de chaque paramètre θ_i. La formule de mise à jour d'Adagrad est :

$$\Delta\theta_{t,i} = -\frac{\eta}{\sqrt{G_{t,ii} + \epsilon}} g_{t,i} \tag{11}$$

qui considère $\epsilon>0$ au dénominateur pour éviter une division par zéro.

Adadelta et al [45] propose une variante d'Adagrad qui vise à réduire l'accumulation de gradients carrés tout le temps pour $G_{ii,}$ et définit à la place une fenêtre moyenne donnée $0\leq\gamma\leq1$ pour pondérer les carrés des gradients actuels et précédents selon :

$$E[g^2]_t = \gamma E[g^2]_{t-1} + (1-\gamma) g_t^2 \tag{12}$$

Ainsi, elle peut être conçue comme l'erreur quadratique moyenne des gradients :

$$RMS[g]_t = \sqrt{E[g^2]_t + \epsilon} \tag{13}$$

Où e > 0 est également inclus pour éviter une division par zéro, étant donné que la mise à jour originale de la formule pour Adadelta est :

$$\Delta\theta_t = -\frac{\eta}{RMS[g]]_t} g_t \tag{14}$$

Pour conserver la même « unité de mesure », le taux d'apprentissage η est remplacé par le RMS, les paramètres sont mis à jour de la manière suivante :

$$\mathrm{RMS}[\Delta\theta]_t = \sqrt{E[\Delta\theta^2]_t + \epsilon} = \sqrt{\gamma E[\Delta\theta^2]_{t-1} + (1-\gamma)\Delta\theta_t^2 + \epsilon} \tag{15}$$

Jusqu'à $t-1$ puisque Δθ est toujours en cours de calcul. Par conséquent, la règle de mise à jour Adadelta est :

$$\Delta\theta_t = -\frac{RMS[\Delta\theta]_{t-1}}{RMS[g]_t} g_t \tag{16}$$

La méthode RMSProp [46] est considéré comme une extension de la méthode d'Adagrad qui maintient un carré moyen mobile des gradients au lieu d'utiliser tout l'historique, et divise le gradient par la racine carrée moyenne de cette moyenne.

Tous les optimiseurs précédents considèrent le gradient gt comme le facteur de mise à jour fondamental qui provient d'une dérivée du premier ordre de la fonction objectif.

Table III.1 Les règles de mise à jour de plusieurs optimiseurs de descente de gradient

Nom	Mettre à jour la règle
SGD	$\Delta\theta_{t,i} = -\eta g_{t,i}$
Adam	$\Delta\theta_{t,i} = -\frac{\eta}{\sqrt{v_t} + \epsilon}\hat{m}_t$
Adadelta	$\Delta\theta_t = -\frac{RMS[\Delta\theta]_{t-1}}{RMS[g]_t} g_t$
Adagrad	$\Delta\theta_{t,i} = -\frac{\eta}{\sqrt{G_{t,ii} + \epsilon}} g_{t,i}$
RMSProp	$\Delta\theta_t = -\frac{\eta}{RMS[g]_t} g_t$

9. Conclusion

Ce chapitre sur le calcul fractionnaire a exploré divers aspects de cette discipline émergente. Après une introduction générale, nous avons défini le calcul fractionnaire et examiné la fonction gamma d'Euler, la fonction bêta, ainsi que les approches d'intégration et de dérivation fractionnaires. Nous avons également abordé l'exponentielle de Mittag-Leffler et les propriétés générales des dérivées fractionnaires. Ces concepts offrent un cadre précieux pour modéliser des phénomènes complexes dans de nombreux domaines scientifiques et techniques.

Dans le chapitre suivant, nous aborderons nos outils et évoquerons nos expériences.

Chapitre IV

Expérimentations

1.Introduction

Dans ce chapitre, nous détaillons le développement d'un système de reconnaissance basé sur l'utilisation des réseaux de neurones, en intégrant un optimiseur fractionnaire pour améliorer les performances du modèle.

Ce chapitre inclut une série d'expériences visant à évaluer l'impact de l'intégration du calcul d'ordre fractionnaire dans les règles de mise à jour des poids lors de l'apprentissage d'un réseau de neurones. Diverses règles de mise à jour des poids seront explorées et testées pour déterminer leur efficacité. Notre travail se base principalement sur celui de [51], tout en explorant plusieurs modifications et pistes de recherche non considérées dans leur étude.

Les résultats de nos expériences seront présentés, accompagnés d'une discussion approfondie sur les conclusions tirées de ces résultats. Tous les programmes ont été développés en Python, en utilisant deux des frameworks les plus utilisés dans le domaine de l'apprentissage profond : TensorFlow et Keras, et exécutés dans l'environnement Google Colab.

La validation de notre méthode sera réalisée en utilisant la base de données MNIST, largement utilisée pour la reconnaissance de chiffres manuscrits.

2. Google Colab

Google Colab (Colaboratory) est un service de Jupyter Notebook hébergé qui ne nécessite aucune configuration et offre un accès gratuit aux ressources informatiques, y compris les GPU et les TPU. Colab est particulièrement bien adapté à l'apprentissage automatique, à la science des données et à l'éducation [47].

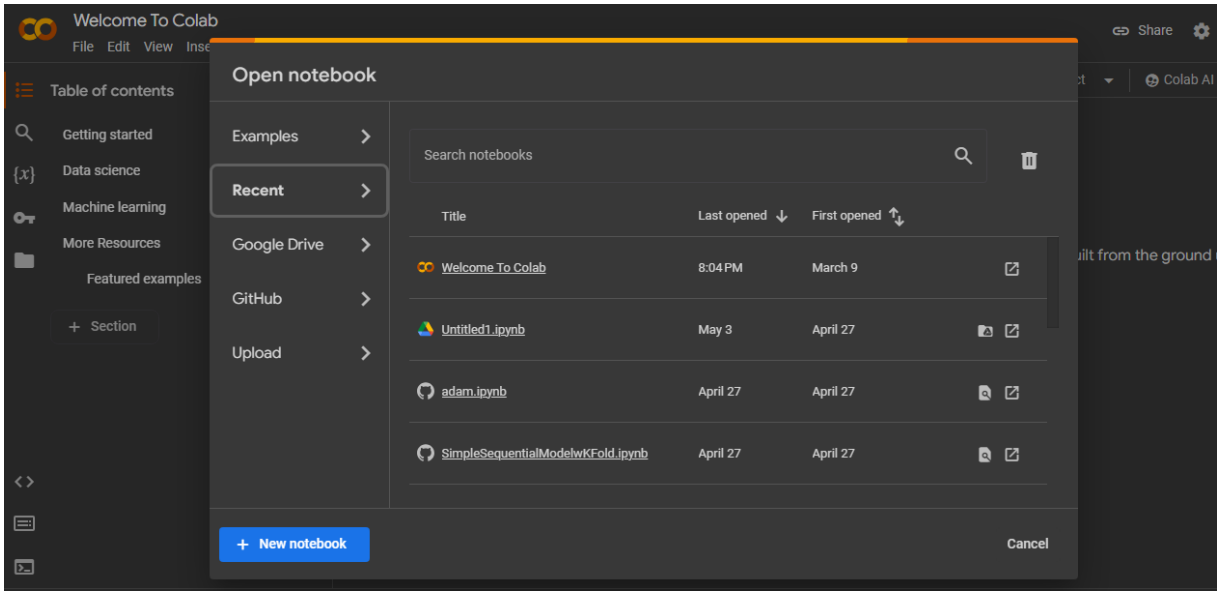

Figure IV.1 Google Colab

3. Tenserflow

TensorFlow est une plateforme open source pour l'apprentissage automatique et l'apprentissage profond développée par Google Brain Team. Elle fournit un écosystème complet d'outils, de bibliothèques et de ressources communautaires facilitant le développement et le déploiement de modèles d'apprentissage automatique. Avec TensorFlow, les utilisateurs peuvent créer et entraîner des réseaux de neurones, effectuer des calculs numériques de manière efficace, et déployer des modèles dans divers environnements, allant des ordinateurs de bureau aux appareils mobiles et aux plateformes cloud. TensorFlow offre flexibilité, évolutivité et facilité d'utilisation, en faisant un choix populaire parmi les chercheurs, les développeurs et les organisations du monde entier.

Figure IV.2 Tensorflow Logo

4. Keras

Keras est une API d'apprentissage profond écrite en Python, capable de s'exécuter sur JAX, TensorFlow, ou PyTorch. Keras se distingue par les caractéristiques suivantes :

- Simplicité : Keras réduit la charge cognitive du développeur, permettant ainsi de se concentrer sur les aspects essentiels du problème sans se perdre dans des détails techniques complexes.
- Flexibilité : Keras adopte le principe de divulgation progressive de la complexité. Les workflows simples sont rapides et faciles à implémenter, tandis que les workflows plus avancés restent accessibles grâce à une approche progressive et claire.
- Puissance : Keras offre des performances et une évolutivité adaptée aux exigences de l'industrie. Il est utilisé par des organisations renommées telles que la NASA, YouTube, et Waymo [48].

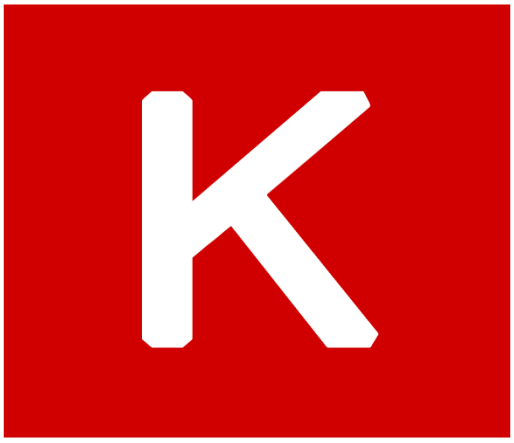

Figure IV.3 : Keras Logo

5. Base de données MNIST

La base de données MNIST (Mixed National Institute of Standards and Technology) est une vaste collection de chiffres manuscrits couramment utilisée dans divers systèmes de traitement d'image.[49] Elle est largement employée pour l'apprentissage et les tests de modèles de reconnaissance de caractères.

La base de données MNIST a été créée en "re-mélangeant" les échantillons provenant des ensembles de données d'origine du NIST. Les images en noir et blanc du NIST ont été normalisées et insérées dans une boîte de 28x28 pixels [50].

Figure IV.4 : Visualisation de l'ensemble de données de MNIST

6. Architecture de Réseau

- **La couche d'entrée (Input Layer)** est définie implicitement par la couche plate, qui prend les images 28x28 et les projettes en un vecteur de 784 dimensions (depuis 28×28=784).
- **Couches cachées (Hidden Layers)** Il y a deux ou plusieurs couches cachées. Toutes les couches cachées utilisent la fonction d'activation ReLu.
- **Couche de sortie (Output Layer)** La couche de production contient 10 neurones correspondant aux 10 classes des chiffres MNIST (0-9), avec une fonction d'activation softmax pour produire une répartition de probabilité sur les classes.

6.1. L'organigramme principale

La méthodologie suivie dans notre travail est basée sur les étapes suivantes :

6.1.1. Importation des bibliothèques : Dans cette étape, nous importons les bibliothèques nécessaires pour notre projet. Celles-ci incluent notamment les bibliothèques to_categorical, Sequential, Dense, Dropout, Flatten, Conv2D, MaxPooling2D, Activation, LeakyReLU, AlphaDropout, et les fonctions associées du backend de Keras.

6.1.2. Chargement de la base de données MNIST : Cette étape consiste à charger les images de chiffres manuscrits ainsi que leurs étiquettes associées.

6.1.3. Prétraitement des données : Avant d'entraîner notre modèle, nous effectuons un prétraitement sur les données. Cela inclut la normalisation des valeurs de pixels (de 0 à 255, ramenées à une échelle de 0 à 1) et la mise en forme des images sous forme de matrices.

6.1.4. Compilation et entraînement : Nous compilons le modèle en utilisant les optimiseurs Fractionnaire et la fonction de perte de catégorisation croisée.

Ensuite, nous procédons à l'entraînement du modèle sur les données d'entraînement pendant 10 époques, en évaluant ses performances à l'aide des données de validation.

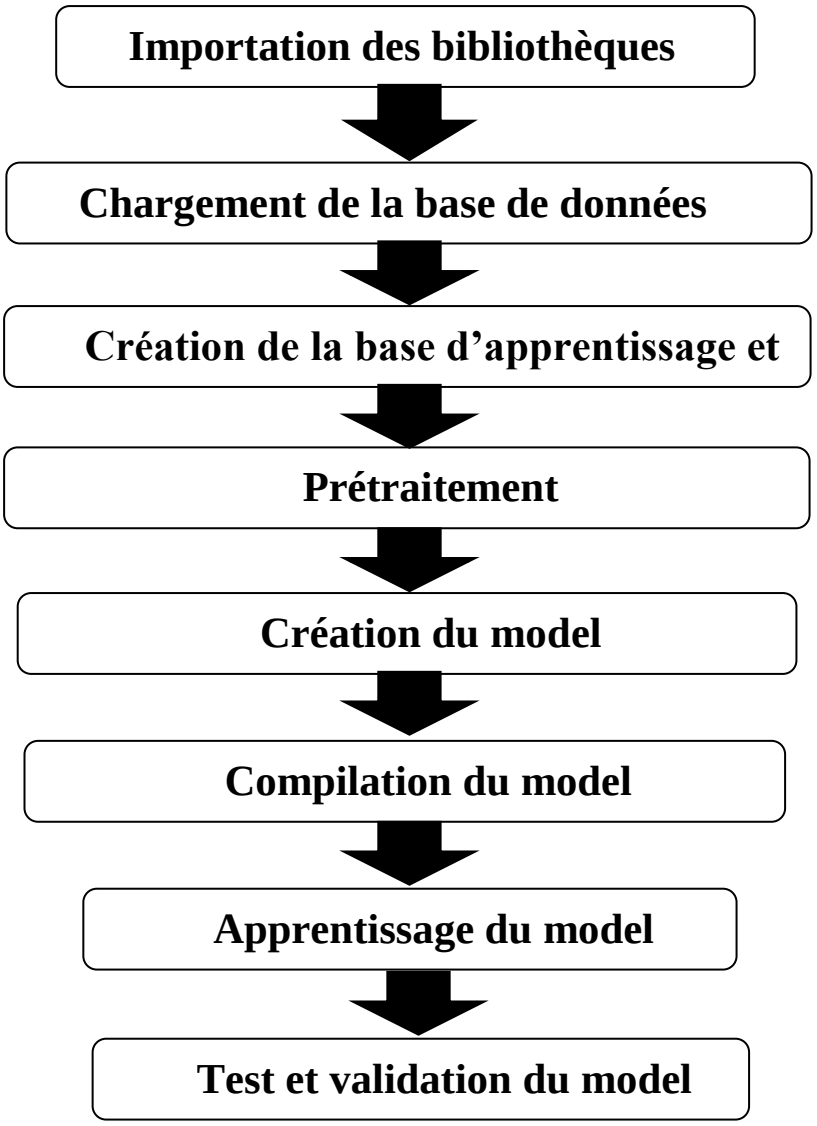

Figure IV.5 : Organigramme principal

Dans ce qui suit, nous allons réaliser une série d'expériences visant à optimiser les performances de notre classificateur. Pour ce faire, nous explorerons deux stratégies distinctes.

La première stratégie consiste à introduire des fonctions d'optimisation fractionnaire en remplacement des fonctions d'optimisation classiques. Cette approche se concentre sur le processus d'apprentissage du modèle, puisque la tâche d'apprentissage repose sur un algorithme qui met à jour les poids du réseau. Cette mise à jour est elle-même basée sur un algorithme d'optimisation, comme décrit dans la section 8 de chapitre 3.

La deuxième stratégie examine l'influence de l'architecture du réseau sur les performances. Nous étudierons comment différentes configurations architecturales peuvent affecter l'efficacité et la précision de notre classificateur.

Ces deux approches nous permettront d'identifier les meilleures pratiques pour améliorer notre modèle, en tenant compte à la fois de l'optimisation des poids et de la structure du réseau.

Toutes les expériences ont été réalisées avec les paramètres suivants :

- Taux d'apprentissage : 0,001
- Époques : 10

Le taux d'apprentissage est un hyperparamètre qui détermine la taille des étapes effectuées par le modèle pour optimiser ses performances. Une époque correspond à un passage complet à travers l'ensemble des données de formation pendant le processus d'apprentissage.

Partie I : Optimisation des performances par les algorithmes d'optimisation fractionnaire

Dans cette partie, nous nous concentrerons sur l'amélioration des performances de notre classificateur en utilisant des algorithmes d'optimisation basée sur le gradient à dérivée fractionnaire. Pour ce faire, nous utiliserons plusieurs algorithmes d'optimisation issus des travaux de la référence [51].

Deux expériences seront réalisées. La première consiste à comparer les performances des optimiseurs fractionnaires entre eux afin de déterminer le meilleur optimisateur. Dans la deuxième expérience, nous comparerons les performances d'un réseau MLP (Multi-Layer Perceptron) avec celles d'un réseau CNN (Convolutional Neural Network).

7. Expérience 1

Dans cette expérience, nous nous intéressons aux méthodes d'optimisation basées sur le calcul fractionnaire, détaillées dans la section 8 chapitre 3 d'optimisation. Nous comparerons les performances du classificateur en utilisant les optimiseurs fractionnaires basés sur le gradient à dérivée fractionnaire suivants :

- FSGD Fractional Stochastic Gradient Descent
- FAdam Fractional Adaptive Moment Estimation
- FAdadelta Fractional Adaptatif delta
- FRMSProp Fractional Root Mean Square
- FAdagrad Fractional Adaptatif grad

Nous allons tout d'abord considérer la même architecture utilisée dans [51], qui comporte des couches entièrement connectées avec deux couches cachées. Le but de cette expérience est de comparer les différents algorithmes d'optimisation afin de déterminer le meilleur optimisateur. Il est à noter que les algorithmes d'optimisation fractionnaire incluent le cas classique, c'est-à-dire le cas entier en prenant l'ordre fractionnaire égal à ν = 1.

7.1 Architecture

Le code Python ci-dessous illustre l'architecture du réseau MLP que nous avons sélectionné :

```
[ ]  # Définir l'architecture du modèle
     model = tf.keras.Sequential([
         tf.keras.layers.Flatten(input_shape=(28 , 28,1)),      # Couche d'entrée (images de 28x28 pixels avec 1 canal de couleur)
         tf.keras.layers.Dense(128, activation='relu'),         # 1ère couche entièrement connectée avec 128 neurones et activation 'relu'
         tf.keras.layers.Dense(64, activation='relu'),          # 2ème couche entièrement connectée avec 64 neurones et activation 'relu'
         tf.keras.layers.Dense(10, activation='softmax')        # Couche de sortie avec 10 neurones et activation 'softmax'
     ])
```

La figure 6 montre la précision de l'optimiseur FAdam en fonction de l'ordre de la dérivée fractionnaire. Dans cette figure, le modèle maintient une grande précision pour toutes les valeurs de ν, avec des précisions généralement supérieures à 97 % pour la plupart des valeurs de ν.

Les précisions médianes les plus élevées sont observées autour de ν=0,4 à ν=0,5, ce qui suggère que cette plage pourrait être optimale pour les performances de l'algorithme FAdam.

La figure 7 montre la précision de l'optimiseur FRMSProp en fonction de l'ordre de la dérivée fractionnaire. Les résultats sont similaires à ceux de l'optimiseur FAdam, avec une précision légèrement inférieure, autour de 96 %, mais des performances globalement solides. Les meilleures précisions sont également observées pour les valeurs de ν=0,4 à ν=0,5.

Dans la figure 8, la précision s'améliore considérablement à mesure que l'ordre de la dérivée fractionnaire ν augmente de 0,1 à environ 1,1. Au-delà de cette valeur, la précision diminue fortement, atteignant environ 79 %, ce qui indique une performance très faible.

Dans la figure 9, la précision de FAdagrad s'améliore généralement à mesure que l'ordre de la dérivée fractionnaire ν augmente de 0,1 à environ 1,6. Au-delà de cette valeur, la précision commence à diminuer, indiquant une performance médiocre.

Dans la figure 10, la précision augmente considérablement entre 0,1 et 1,1, atteignant 89 %. Elle continue ensuite de s'améliorer, bien que plus lentement, jusqu'à 1,7, où elle atteint 92,3 %. Au-delà de cette valeur, la précision diminue, indiquant une performance médiocre, similaire à celle de FAdagrad.

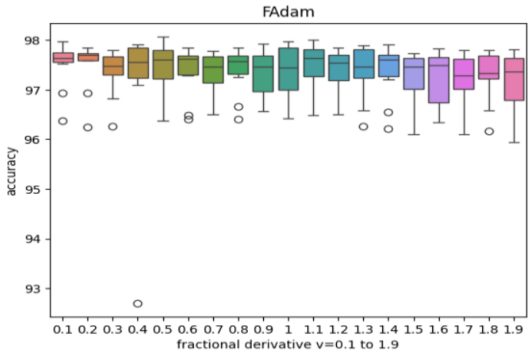

Figure IV.6 : Precision d'optimiseur FAdam

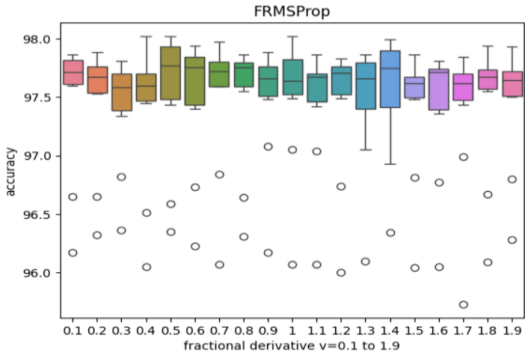

Figure IV.7 : Precision d'optimiseur FRMSProp

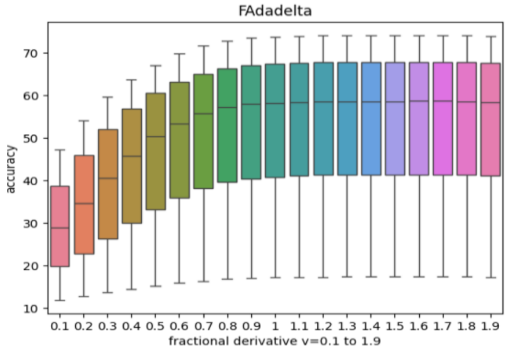

Figure IV.8 : Precision d'optimiseur Fadadelta

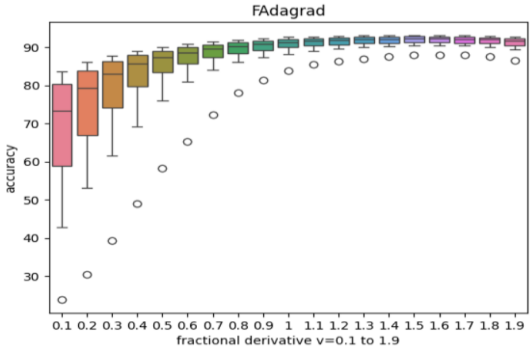

Figure IV.9 : Precision d'optimiseur Fadagrad

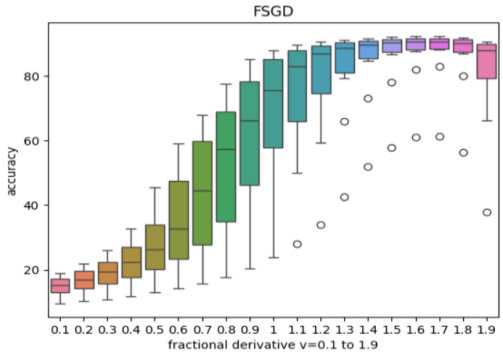

Figure IV.10 : Precision d'optimiseur FSGD

Les matrices de confusion pour les différents optimiseurs utilisés sont représentées dans les figures 11-15.

Les Figure 11-15 montrent que les algorithmes FAdam et FRMSprop offrent les meilleures performances globales, avec un nombre réduit d'erreurs de classification. Les erreurs persistantes se concentrent principalement entre les chiffres 4 et 9, ainsi que 3 et 5.

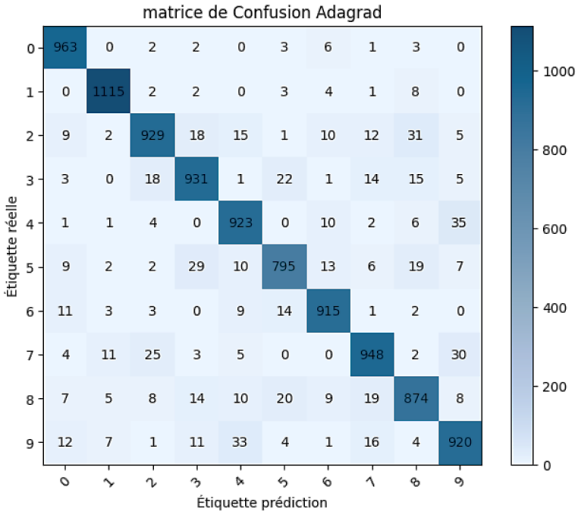

Figure IV.11 : Matrice de confusion FADagrad

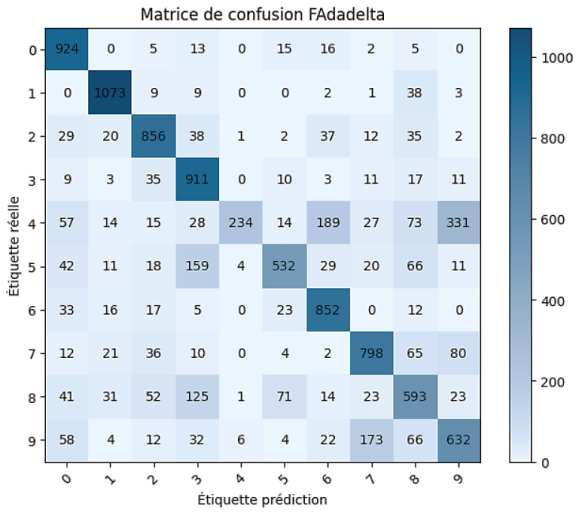

Figure IV.12 : matrice de confusion FAdadelta

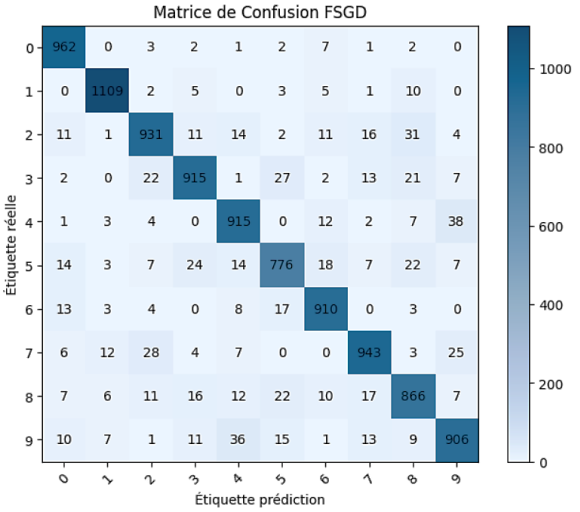

Figure IV.13 : matrice de confusion FSGD

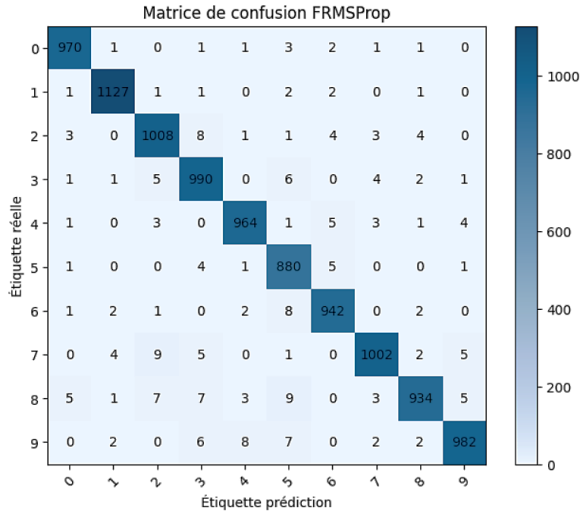

Figure IV.14 : matrice de confusion FRMSprop

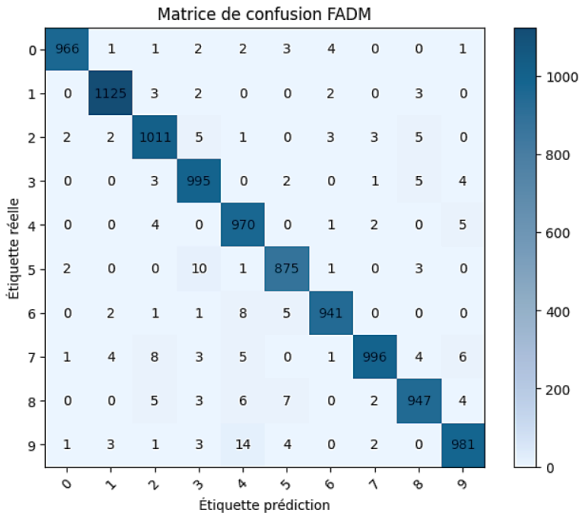

Figure IV.15 : matrice de confusion FAdam

Figure 11-15 Les matrices de confusion pour les différents optimiseurs utilisés

Le tableau 1 donne la précision des optimiseurs fractionnaires en fonction de l'ordre de la dérivée.

Table IV.1 calcul de la précision des optimiseurs fractionnaire en fonction de l'ordre de la dérivée

V / Op	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0	1.1	1.2	1.3	1.4	1.5	1.6	1.7	1.8	1.9
FSGD	18.9	21.8	26.0	32.7	45.5	59.1	67.9	77.5	85.1	**87.9**	89.5	90.5	91.1	91.6	92.0	92.2	**92.3**	91.8	90.6
FAdam	97.7	97.6	97.6	97.8	**98.0**	97.6	97.6	97.8	97.3	**97.8**	97.8	97.75	97.8	97.6	97.4	97.8	97.6	97.7	97.7
FAdagrad	80.3	86.0	87.7	88.9	89.9	90.7	91.4	91.8	92.3	**92.6**	92.7	92.9	93.0	93.1	**93.1**	93.1	93.0	92.9	92.6
FAdadelta	47.3	65.0	67.8	73.6	71.0	75.1	77.5	73.4	74.4	**78.3**	77.2	78.4	**79.5**	78.8	75.4	76.0	77.7	79.1	75.6
FRMSProp	97.8	97.7	97.6	97.7	**97.9**	97.8	97.7	97.9	97.7	**97.6**	97.6	97.7	97.8	97.8	97.7	97.7	97.7	97.8	97.8

Après avoir testé plusieurs optimiseurs avec différentes dérivées fractionnaires, nous avons constaté que les optimiseurs fractionnaires offrent de meilleures performances par rapport à leurs homologues classiques. Cette supériorité s'explique par le fait que les optimiseurs fractionnaires fournissent un degré de liberté supplémentaire pour l'optimisation des paramètres du réseau neuronal.

8. Experience 2

Dans cette expérience, nous allons maintenir les mêmes conditions que dans l'expérience précédente et tester l'influence du type de réseau sur les performances globales du classificateur. Pour cela, nous utiliserons un réseau de neurones convolutionnel (CNN) à la place du réseau de neurones multicouches entièrement connecté. Le but est de comparer ces architectures et de déterminer laquelle offre les meilleures performances.

8.1 Architecture

Le code Python ci-dessous illustre l'architecture du réseau CNN que nous avons sélectionné :

```
# Définir l'architecture du modèle avec 2 couches cachées
input_layer = Input(shape=(28, 28, 1))                    # Couche d'entrée

# 1ère Couche Cachée : Couche de convolution avec Dropout
x = Conv2D(32, (3, 3), activation='relu')(input_layer)    # Couche de convolution
x = MaxPooling2D((2, 2))(x)                               # Couche de pooling (non comptée comme une couche cachée)
x = Dropout(0.25)(x)                                      # Couche Dropout avec un taux de dropout de 25%

# Flattening de la carte des caractéristiques
x = Flatten()(x)

# 2ème Couche Cachée : Couche entièrement connectée avec Dropout
x = Dense(128, activation='relu')(x)                      # Couche entièrement connectée
x = Dropout(0.5)(x)                                       # Couche Dropout avec un taux de dropout de 50%

# Couche de sortie
output_layer = Dense(10, activation='softmax')(x)         # Couche de sortie

# Créer le modèle
model = Model(inputs=input_layer, outputs=output_layer)
```

Dans ce qui suit, nous allons analyser la variation de la précision en fonction de l'ordre de la dérivée fractionnaire. Les figures 16, 17, 18, 19 et 20 montrent la précision en fonction de l'ordre fractionnaire pour les optimiseurs Fadadelta, Fadagrad, Fadam, FRMSProp et FSGD respectivement.

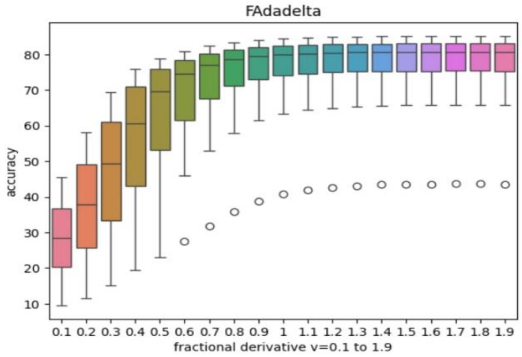

Figure IV.16 : Precision d'optimiseur FAdadelta

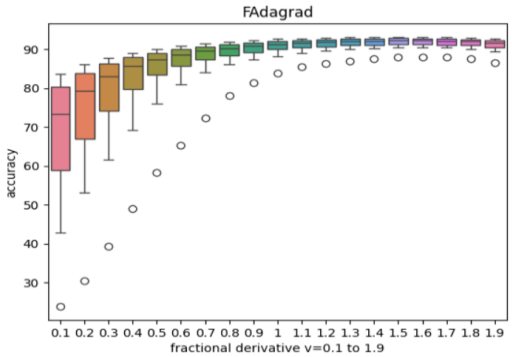

Figure IV.17 : Precision d'optimiseur FAdagrad

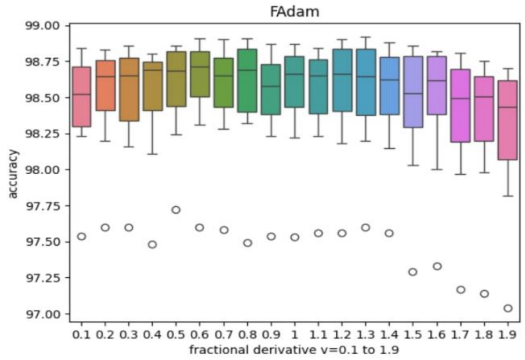

Figure IV.18 : Precision d'optimiseur FAdam

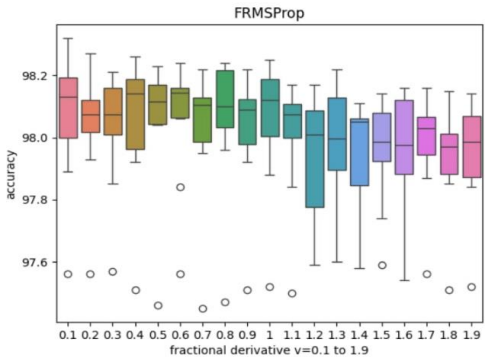

Figure IV.19 : Precision d'optimiseur FRMSProp

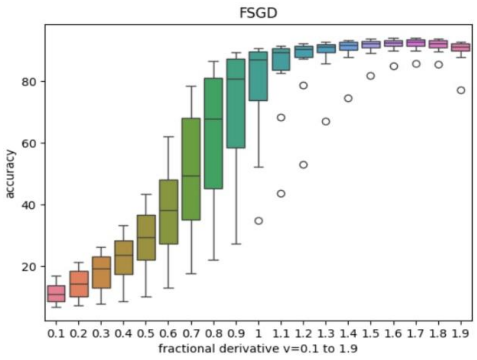

Figure IV.20 : Precision d'optimiseur FSGD

Dans la figure (16), la précision s'améliore considérablement à mesure que l'ordre de la dérivée fractionnaire v augmente de 0,1 à environ 1,1. Au-delà de v = 1,1, la précision se stabilise et reste relativement élevée et stable jusqu'à la fin. La plage optimale pour v en

termes de maximisation de la précision se situe entre environ 1,5 et 1,9. Dans cette plage, le modèle atteint une précision élevée et stable.

Dans figure 17, la précision s'améliore généralement à mesure que l'ordre de la dérivée fractionnaire v augmente de 0,1 à environ 1,6. Au-delà de cette valeur, la précision diminue légèrement et devient erratique. La plage optimale pour v en termes de maximisation de la précision se situe entre environ 1,4 et 1,6. Dans cette plage, le modèle atteint une précision élevée et stable.

Dans la figure 18, le modèle maintient une grande précision sur toutes les valeurs de v, avec des précisions généralement supérieures à 98,00 %. Les précisions médianes les plus élevées sont observées autour de v = 1,1 à v = 1,3, ce qui suggère que cette plage pourrait être optimale pour les performances de l'algorithme FAdam.

Idem avec l'optimiseur Adam, Le modèle affiche une grande précision sur la plupart des valeurs de v, généralement supérieures à 98,0 %, mais avec des variations notables. Les précisions médianes les plus élevées sont observées autour de v = 0,1 à v = 0,4, ce qui indique que cette plage peut être optimale pour l'algorithme FRMSProp.

Sur la figure 20, une nette tendance à la hausse en termes de précision est observée à mesure que v passe de 0,1 à environ 1,6. Au-delà de ce point, la précision semble se stabiliser à des valeurs élevées, proches de 96 %. L'algorithme FSGD fonctionne mal aux faibles valeurs de v, s'améliore considérablement à mesure que v augmente, et atteint une précision stable et élevée pour les valeurs de v d'environ 1,1 à 1,6.

Les figues 21-25 montre les résultats des matrices de confusion pour différents optimiseurs utilisés.

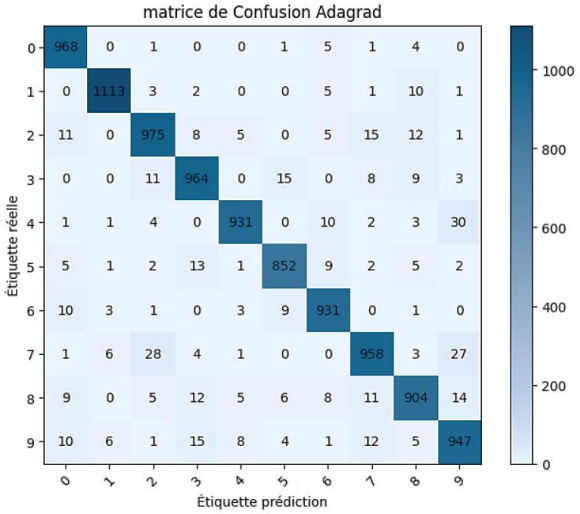

Figure IV.21 : matrice de confusion FAdagrad

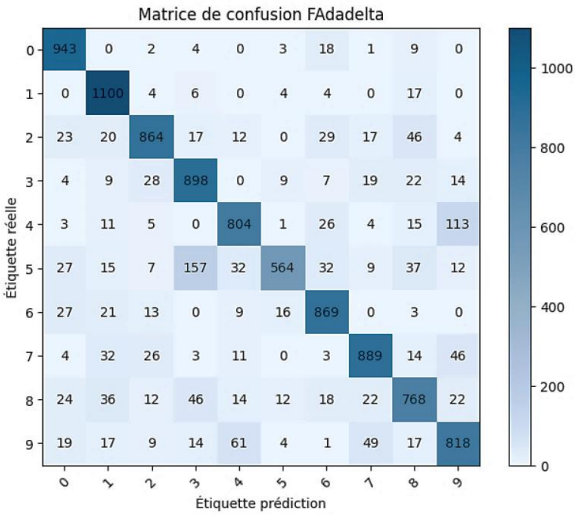

Figure IV.22 : matrice de confusion FAdadelta

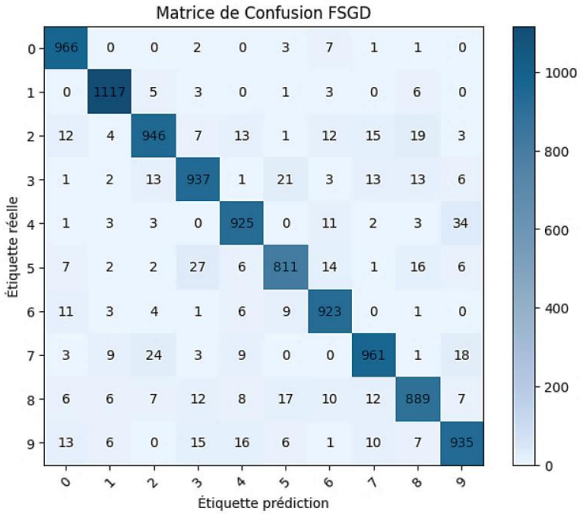

Figure IV.23 : matrice de confusion FSGD

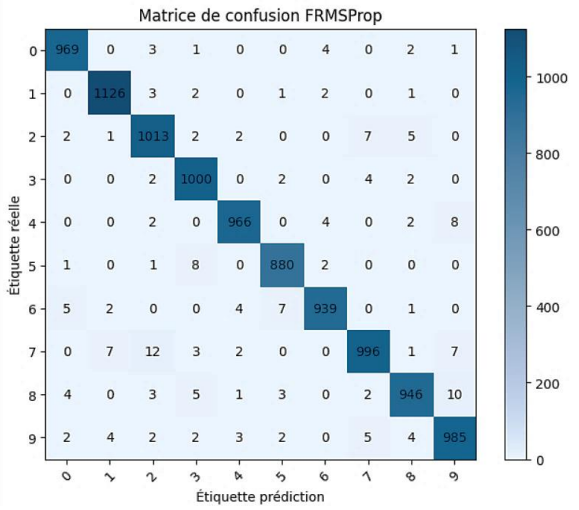

Figure IV.24 : matrice de confusion FRMSprop

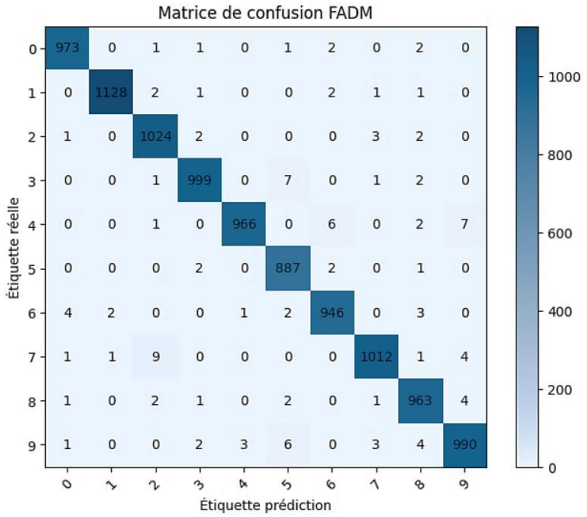

Figure IV.25 : matrice de confusion FAdam

Les Figures 21-25 Les matrices de confusion pour les différents optimiseurs utilisés

Selon les Figures 21-25, FAdam est l'optimisateur le plus performant, démontrant une haute précision et un faible taux d'erreurs comparé à FAdadelta, FSGD, FRMSprop et FAdagrad.

Ses performances supérieures en termes de précision et de sensibilité le rendent particulièrement efficace pour diverses tâches d'apprentissage machine. Bien que les confusions entre les chiffres 7 et 2, ainsi que 4 et 9, soient récurrentes, elles sont moins fréquentes avec FAdam.

Le tableau 2 donne la précision des optimiseurs fractionnaires en fonction de l'ordre de la dérivée dans le cas d'une architecture de réseau de neurones CNN, dans le cas de deux couches cachées.

Tableau IV.2: Calcul de la précision des optimiseurs fractionnaires en fonction de l'ordre de la dérivée dans le cas d'une architecture de réseau de neurones CNN.

V / Op	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0	1.1	1.2	1.3	1.4	1.5	1.6	1.7	1.8	1.9
FSGD	16.9	21.4	26.4	33.3	43.5	62.2	78.4	86.6	89.3	**90.6**	91.5	92.2	92.8	93.4	93.7	**94.1**	94.0	93.8	92.7
FAdam	98.8	98.8	98.7	98.7	98.8	98.8	98.8	98.9	98.8	**98.8**	98.7	98.8	**98.8**	98.8	98.8	98.8	98.8	98.7	98.7
FAdagrad	83.8	86.9	88.5	89.8	90.8	91.5	92.1	92.5	92.8	**93.2**	93.4	93.7	93.9	94.1	94.2	**94.3**	94.3	94.3	94.0
FAdadelta	45.4	58.1	69.4	75.8	78.7	80.9	82.4	83.3	84.0	**84.4**	84.7	84.9	85.0	85.0	85.1	85.1	**85.1**	85.1	85.1
FRMSProp	98.1	98.0	98.0	98.1	98.1	98.1	98.2	98.2	98.1	**98.1**	98.1	97.9	**98.2**	98.0	97.9	98.1	98.0	97.9	97.9

Selon les résultats obtenus, nous avons observé que les optimiseurs fractionnaires surpassent en performances les optimiseurs classiques, même après modification de l'architecture. De plus, nous avons constaté que les architectures utilisant des couches convolutives (réseaux de neurones convolutifs) surpassent les réseaux de neurones multicouches entièrement connectés.

Partie II : Optimisation des performances par le choix de l'architecture du réseau de neurones

Dans cette deuxième partie nous allons examiner l'influence de l'architecture du réseau sur les performances de classification. Nous étudierons comment différentes configurations architecturales peuvent affecter l'efficacité et la précision de notre classificateur.

9. Expérience 3

Dans cette expérience, nous voulons étudier l'influence de l'ajout d'une 3ème et 4ème couche au réseau CNN sur l'amélioration de la précision de classification. Pour ce faire, nous

proposons deux architectures : la première comportant trois couches convolutives, tandis que la deuxième en comporte quatre. Nous maintiendrons les mêmes paramètres que dans les expériences précédentes. Ceci nous permettra d'évaluer l'impact de l'ajout de ces couches convolutives.

9.1 Architecture avec trois couches convolutive

Le code Python ci-dessous illustre l'architecture du réseau CNN utilisé :

```
# Définir l'architecture du modèle avec 3 couches cachées
input_layer = Input(shape=(28, 28, 1))                    # Couche d'entrée

# 1ère Couche Cachée : Couche de convolution avec Dropout
x = Conv2D(32, (3, 3), activation='relu')(input_layer)    # 1ère couche de convolution
x = MaxPooling2D((2, 2))(x)                               # Couche de pooling
x = Dropout(0.25)(x)                                      # Couche Dropout avec un taux de dropout de 25%

# 2ème Couche Cachée : Couche de convolution avec Dropout
x = Conv2D(64, (3, 3), activation='relu')(x)              # 2ème couche de convolution
x = MaxPooling2D((2, 2))(x)                               # Couche de pooling
x = Dropout(0.25)(x)                                      # Couche Dropout avec un taux de dropout de 25%

# Flattening de la carte des caractéristiques
x = Flatten()(x)

# 3ème Couche Cachée : Couche entièrement connectée avec Dropout
x = Dense(128, activation='relu')(x)                      # Couche entièrement connectée
x = Dropout(0.5)(x)                                       # Couche Dropout avec un taux de dropout de 50%

# Couche de sortie
output_layer = Dense(10, activation='softmax')(x)         # Couche de sortie
```

Dans ce qui suit, nous allons analyser la variation de la précision de classification en fonction de l'ordre de la dérivée fractionnaire. Les figures 26, 27, 28, 29 et 30 montrent la précision en fonction de l'ordre fractionnaire pour les optimiseurs Fadagrad, Fadam, FSGD, FRMSProp et Fadadelta respectivement, dans le cas d'une architecture à 3 couches convolutives.

La figure 26 montre que la précision augmente en fonction de la dérivée fractionnaire *v*. Elle augmente de 0,1 à environ 1,6 avec une précision de 95 %, puis diminue pour les valeurs de *v* supérieures à 1,6.

Sur la figure 27, on remarque une très grande précision pour toutes les valeurs de *v* par rapport à FAdam avec 2 couches cachées, atteignant 99,2 % et plus pour toutes les valeurs de dérivées fractionnaires.

Sur la figure 28, on remarque une augmentation de la précision de 94 % par rapport à l'architecture comportant 2 couches de convolution. La précision atteint 95 % dans cette nouvelle architecture.

Concernant la figure 29, la précision est généralement élevée pour tous les ordres de dérivées fractionnaires, se situant généralement entre 98,6 % et 99,0 %. La performance optimale se trouve autour de 0,3 et 0,9, où la précision est à la fois élevée et stable.

La figure 30 montre que l'optimiseur FAdelta présente une baisse significative de performances, avec une perte de précision d'environ 10 % ou plus dans certaines dérivées fractionnaires.

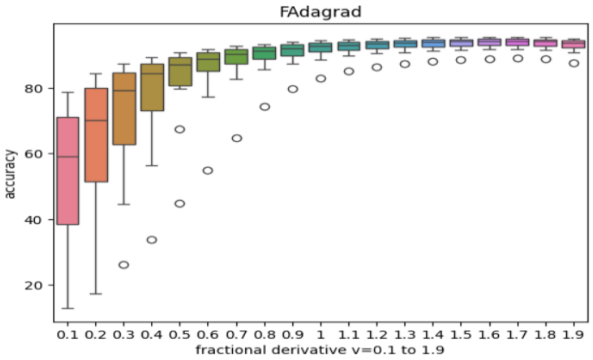

Figure IV.26 : Precision d'optimiseur FAdagrad

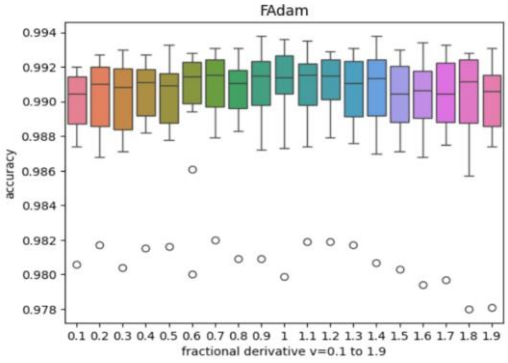

Figure IV.27 : Precision d'optimiseur FAdam

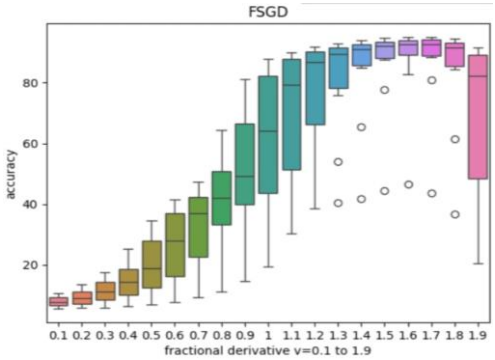

Figure IV.28 : Precision d'optimiseur FSGD

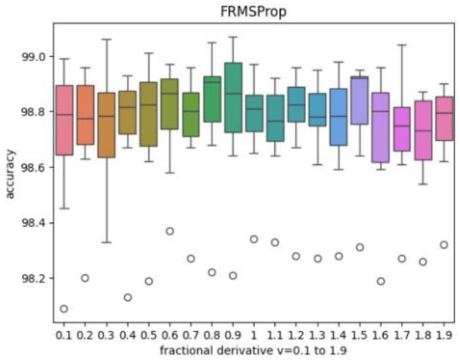

Figure IV.29 : Precision d'optimiseur FRMSProp

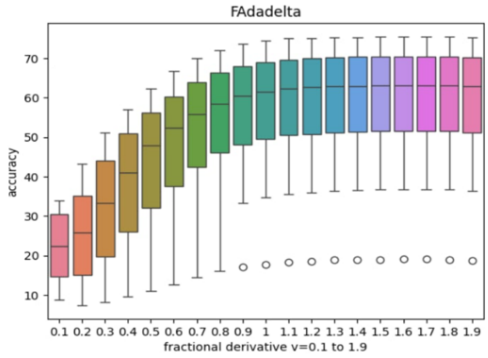

Figure IV.30 : Precision d'optimiseur Fadadelta

Les figues 31-35 montre les résultats des matrices de confusion pour différents optimiseurs utilisés.

Les figures 31 à 35, montrent des performances supérieures par rapport aux résultats de l'expérience précédente. Les erreurs persistent principalement entre les chiffres 4 et 9, ainsi que 8 et 3, mais elles sont moins fréquentes avec FAdam.

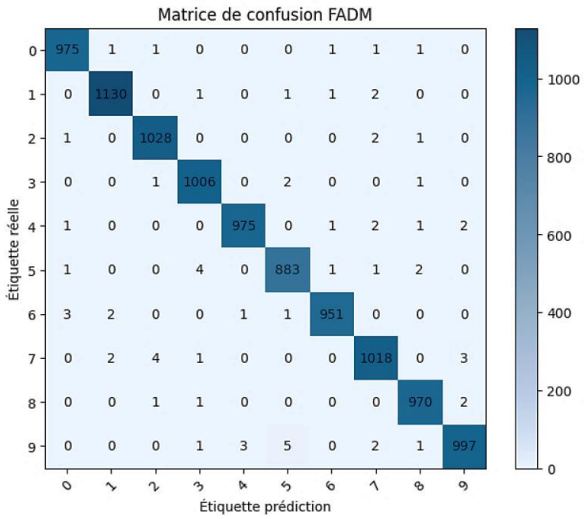

Figure IV.31 : matrice de confusion FADM

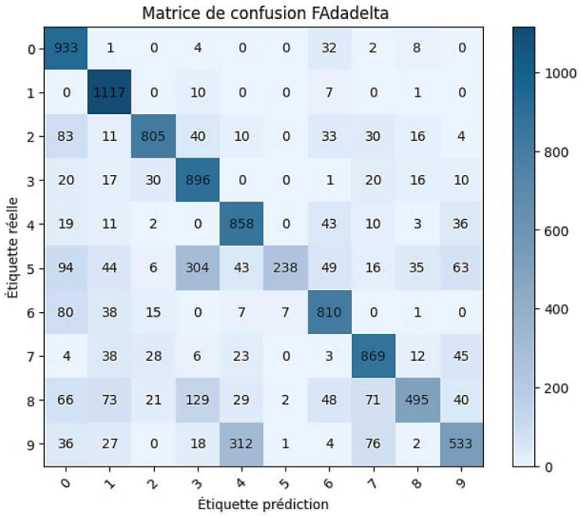

Figure IV.32 : matrice de confusion FAdadelta

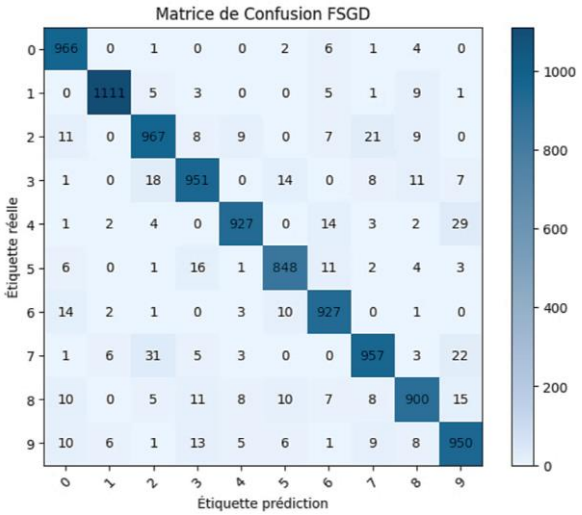

Figure IV.33 : matrice de confusion FSGD

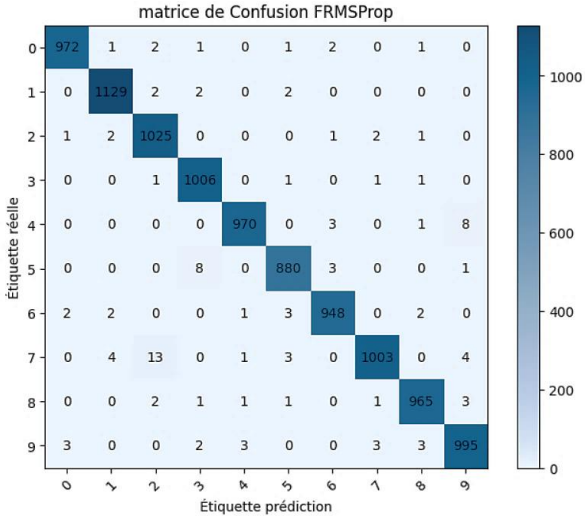

Figure IV.34 : matrice de confusion FRMSprop

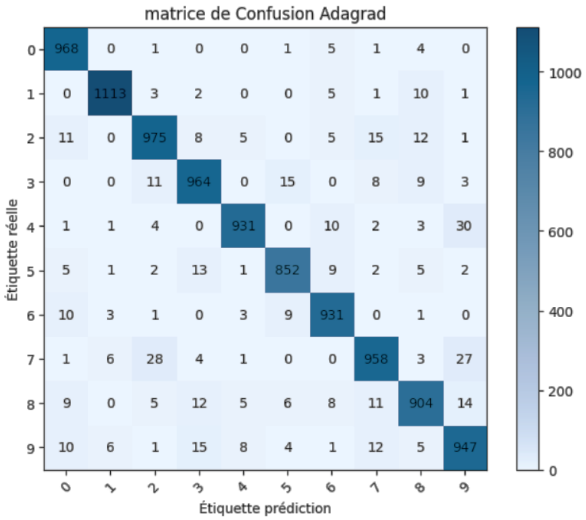

Figure IV.35 : matrice de confusion FAdagrad

Les Figures 31-35 Les matrices de confusion pour les différents optimiseurs utilisés

Le tableau 3 donne la précision des optimiseurs fractionnaires en fonction de l'ordre de la dérivée dans le cas d'une architecture de réseau de neurones CNN, dans le cas de trois couches cachées.

Tableau IV.3: Calcul de la précision des optimiseurs fractionnaires en fonction de l'ordre de la dérivée dans le cas d'une architecture de réseau de neurones CNN.

V / Op	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0	1.1	1.2	1.3	1.4	1.5	1.6	1.7	1.8	1.9
FSGD	10.5	13.4	17.4	25.2	34.4	41.3	74.4	74.3	81.2	**87.7**	90.0	91.9	93.0	94.0	94.6	95.0	**95.0**	94.4	91.4
FAdam	99.2	99.2	99.3	99.2	99.3	99.2	99.2	99.3	99.2	**99.3**	99.3	99.2	99.2	99.3	99.3	99.3	**99.3**	99.2	99.3
FAdagrad	78.6	84.4	87.3	89.4	90.8	91.8	92.7	93.3	93.9	**94.4**	94.7	95.0	95.2	95.3	95.4	95.4	**95.4**	95.3	95.0
FAdadelta	33.8	43.2	51.2	57.0	62.3	66.7	70.0	72.1	73.6	**74.5**	75.0	75.1	75.3	75.4	75.4	75.5	**75.5**	75.5	75.3
FRMSProp	98.9	98.6	98.6	98.8	98.6	98.9	98.7	98.9	98.8	**98.7**	98.6	98.8	98.7	98.7	**98.9**	98.6	98.6	98.5	98.8

L'analyse du resultats du tableau 3 et du tableau 2, montre que une architecture à trois couches cachées surpasse une architecture à deux couches cachées sur tous les optimiseurs fractionnaires, à l'exception du cas de Fadadelta.

Bien que 3 couches cachées puissent offrir plus de capacité, cette complexité supplémentaire peut également introduire des défis tels que des problèmes de gradient et de sensibilité aux hyperparamètres, ce qui pourrait expliquer pourquoi l'optimiseur FadaDelta fonctionne mieux avec 2 couches cachées dans ce contexte.

9.2 Architecture avec quatre couches cachées

Le code Python ci-dessous illustre l'architecture du réseau CNN utilisé :

```
# Définir l'architecture du modèle avec 4 couches cachées
input_layer = Input(shape=(28, 28, 1))                    # Couche d'entrée

x = Conv2D(32, (3, 3), activation='relu')(input_layer)    #  couche de convolution
x = MaxPooling2D((2, 2))(x)                               #  couche de pooling
x = Dropout(0.2)(x)                                       #  couche Dropout avec un taux de dropout de 20%

x = Conv2D(64, (3, 3), activation='relu')(x)              # couche de convolution
x = MaxPooling2D((2, 2))(x)                               #  couche de pooling
x = Dropout(0.2)(x)                                       #  couche Dropout avec un taux de dropout de 20%

x = Conv2D(128, (3, 3), activation='relu')(x)             #  couche de convolution
x = MaxPooling2D((2, 2))(x)                               #  couche de pooling
x = Dropout(0.2)(x)                                       #  couche Dropout avec un taux de dropout de 20%

x = Flatten()(x)                                          # Aplatissement de la carte des caractéristiques

x = Dense(128, activation='relu')(x)                      # Couche entièrement connectée
x = Dropout(0.2)(x)                                       # Couche Dropout avec un taux de dropout de 20%

output_layer = Dense(10, activation='softmax')(x)         # Couche de sortie

model = Model(inputs=input_layer, outputs=output_layer)   # Créer le modèle
```

Les figures 36, 37, 38, 39 et 40 montrent la précision en fonction de l'ordre fractionnaire pour les optimiseurs Fadadelta, Fadagrad, Fadam, FRMSProp et FSGD respectivement.

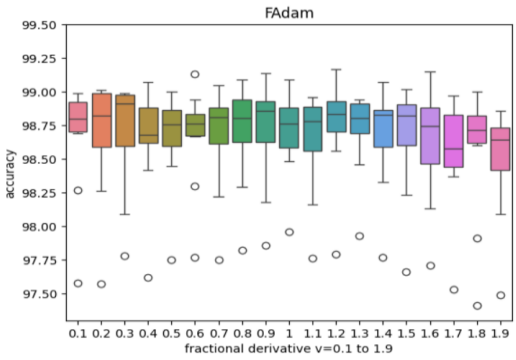

Figure IV.36 : Precision d'optimiseur FAdam

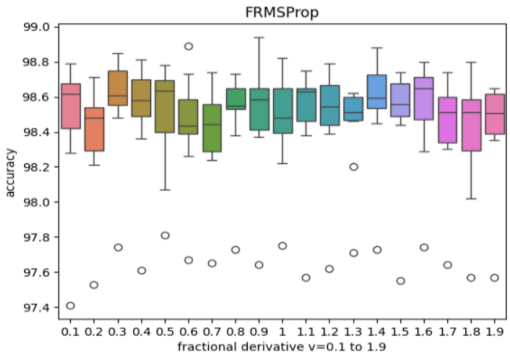

Figure IV.37 : Precision d'optimiseur FRMSProp

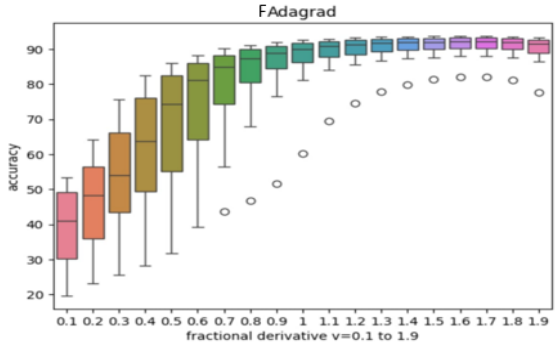

Figure IV.38 : Precision d'optimiseur FAdagrad

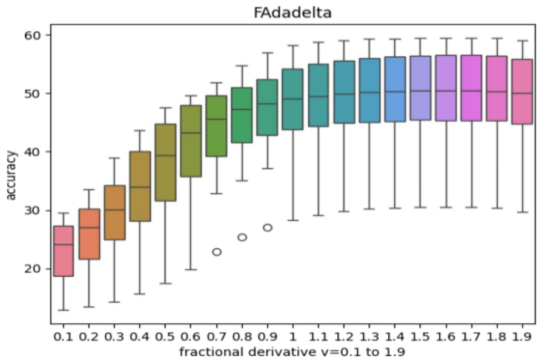

Figure IV.39 : Precision d'optimiseur FAdadelta

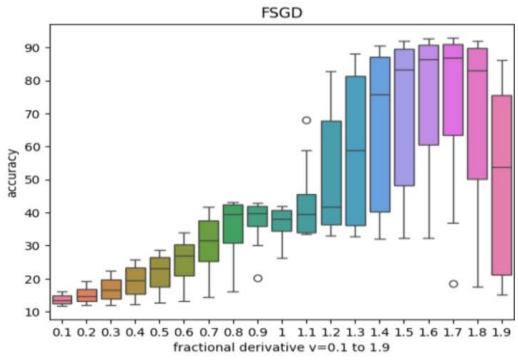

Figure IV.40 : Precision d'optimiseur FSGD

Dans les figures (36-40), on constate une baisse de performances de tous les optimiseurs de cette architecture par rapport à l'architecture précédente d'environ 0,3% à 9%.

Les figues 41-45 montre les résultats des matrices de confusion pour différents optimiseurs utilisés.

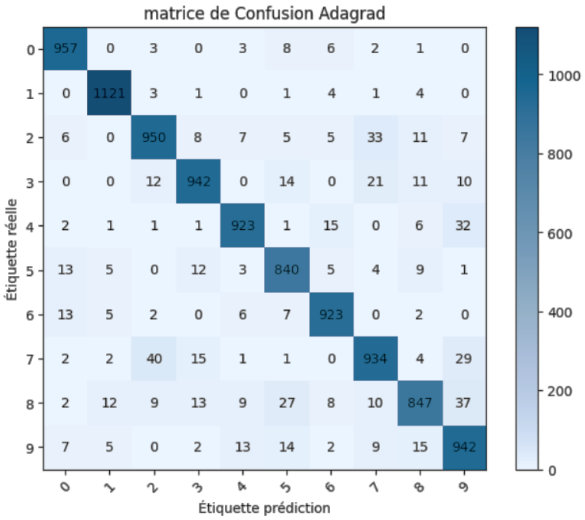

Figure IV.41 : matrice de confusion FAdagrad

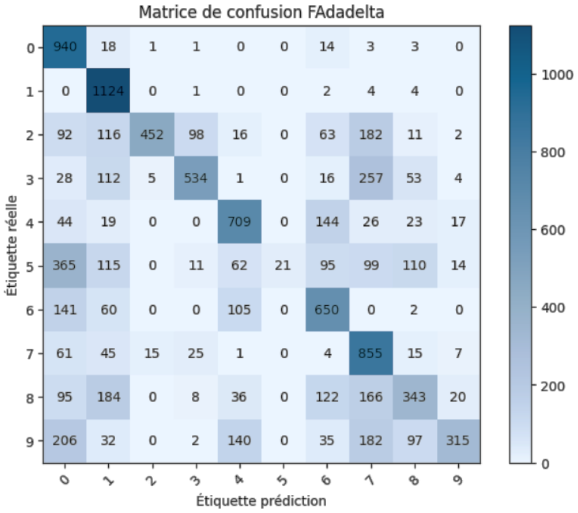

Figure IV.42 : matrice de confusion FAdadelta

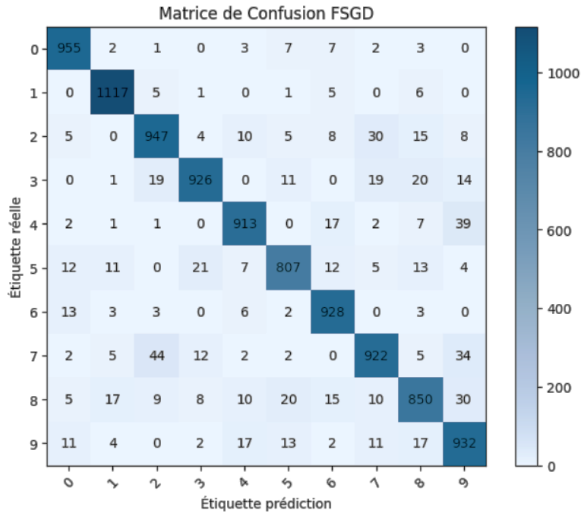

Figure IV.43 : matrice de confusion FSGD

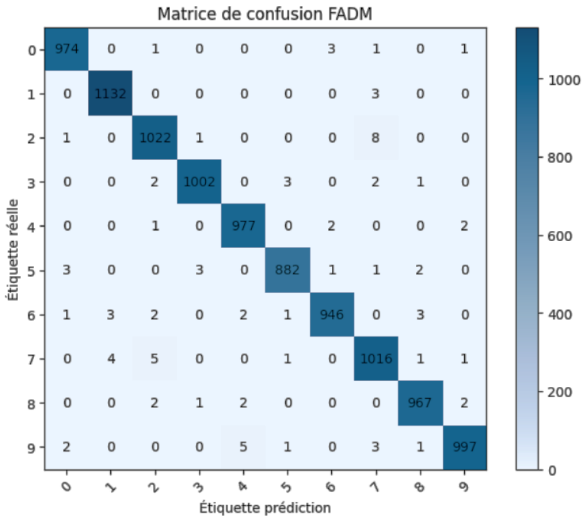

Figure IV.44 : matrice de confusion FAdam

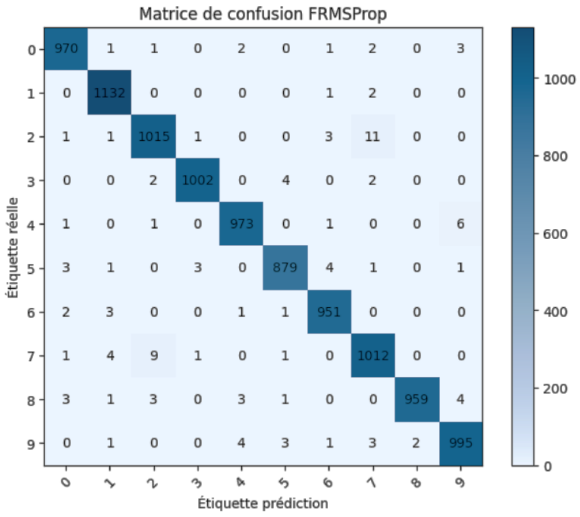

Figure IV.45 : matrice de confusion FRMSprop

Figure 41-45 Les matrices de confusion pour les différents optimiseurs utilisés

L'analyse des matrices de confusion des différents optimisateurs (Figure 41-45), indiquent que FAdam reste le plus performant avec un nombre réduit d'erreurs de classification. Les confusions entre les chiffres 3 et 5, et 7 et 1 sont les plus notables mais significativement réduites avec FAdam.

Le Tableau 4 donne la précision des optimiseurs fractionnaires en fonction de l'ordre de la dérivée dans le cas d'une architecture de réseau de neurones CNN dans le cas de quatre couches cachées.

Tableau IV.4: Calcul de la précision des optimiseurs fractionnaires en fonction de l'ordre de la dérivée dans le cas d'une architecture de réseau de neurones CNN.

V / Op	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0	1.1	1.2	1.3	1.4	1.5	1.6	1.7	1.8	1.9
FSGD	16.2	19.2	22.4	25.8	28.7	34.0	41.6	42.1	35.5	**38.4**	67.9	82.6	88.0	90.4	91.9	92.7	**92.9**	91.8	86.1
FAdam	98.9	98.9	99.0	99.0	99.0	99.1	99.0	99.0	99.1	**98.9**	98.8	99.0	98.9	98.8	98.9	**99.1**	98.9	99.0	98.8
FAdagrad	53.3	64.1	75.7	82.4	86.0	88.3	90.1	91.1	92.0	**92.6**	92.9	93.3	93.4	93.6	93.7	**93.7**	93.7	93.6	93.2
FAdadelta	29.5	33.4	39.0	43.6	47.5	49.6	51.8	54.7	56.9	**58.1**	58.7	58.9	59.2	59.3	59.4	**59.4**	59.4	59.4	59.0
FRMSProp	98.7	98.5	98.5	98.4	98.7	98.4	98.4	98.5	98.5	**98.4**	98.6	98.5	98.5	**98.8**	98.5	98.7	98.6	98.0	98.4

L'analyse des résultats du tableau 2, tableau 3 et tableau 4, montre que FAdam avec une architecture à trois couches cachées surpasse tous les autres optimiseurs dans toutes les configurations.

10. Conclusion

Dans ce chapitre, nous avons présenté une série d'expériences visant à évaluer l'impact des optimiseurs fractionnaires sur les performances d'un modèle de reconnaissance de chiffres manuscrits basé sur des réseaux de neurones. Les résultats obtenus démontrent la supériorité des optimiseurs fractionnaires par rapport à leurs homologues classiques, offrant un degré de liberté supplémentaire pour l'optimisation des paramètres du réseau.

Parmi les différents optimiseurs fractionnaires étudiés, l'optimiseur FAdam s'est révélé être le plus performant dans la plupart des configurations, atteignant des taux de précision supérieurs à 99% pour la classification des chiffres de la base de données MNIST. Nous avons également observé que les réseaux de neurones convolutionnels (CNN) surpassent les

réseaux de neurones multicouches entièrement connectés (MLP) en termes de précision de classification.

De plus, nos expériences ont mis en évidence que l'ajout de couches convolutives supplémentaires peut améliorer les performances jusqu'à un certain point, mais un nombre excessif de couches peut conduire à une dégradation des résultats. L'architecture optimale identifiée dans cette étude est un réseau CNN comportant trois couches convolutives, en combinaison avec l'optimiseur fractionnaire FAdam.

Ces résultats prometteurs ouvrent de nouvelles perspectives pour l'utilisation des optimiseurs fractionnaires dans d'autres applications de l'apprentissage profond, offrant un potentiel d'amélioration des performances par rapport aux méthodes classiques. Cependant, des travaux supplémentaires seront nécessaires pour explorer davantage les propriétés et les implications de ces optimiseurs fractionnaires, ainsi que pour étendre leur application à des problèmes plus complexes et à d'autres domaines.

Conclusion générale

Le présent travail a exploré de manière approfondie la reconnaissance des chiffres manuscrits en mettant l'accent sur l'utilisation de l'apprentissage profond et du calcul d'ordre fractionnaire. Notre objectif principal était d'évaluer l'impact de l'intégration du calcul fractionnaire dans les règles de mise à jour des poids lors de l'entraînement des réseaux de neurones.

Une méthodologie rigoureuse a été suivie, impliquant l'importation des bibliothèques nécessaires, le chargement de la base de données MNIST, le prétraitement des données, la compilation et l'entraînement des modèles. Une série d'expériences a été menée pour comparer les performances de différents optimiseurs fractionnaires basés sur le gradient à dérivée fractionnaire.

Les résultats obtenus démontrent la supériorité des optimiseurs fractionnaires par rapport à leurs homologues classiques. Cette amélioration des performances s'explique par le degré de liberté supplémentaire offert par les dérivées fractionnaires dans l'optimisation des paramètres du réseau.

De plus, nos expériences ont révélé que les architectures utilisant des couches convolutives, telles que les réseaux de neurones convolutionnels (CNN), surpassent les réseaux de neurones multicouches entièrement connectés. L'ajout de couches supplémentaires a également été étudié, mettant en évidence l'amélioration de la précision avec trois couches cachées, bien que la complexité accrue puisse introduire des défis tels que les problèmes de gradient.

En particulier, l'optimiseur FAdam couplé à une architecture CNN à trois couches cachées s'est avéré offrir les meilleures performances globales pour la reconnaissance des chiffres manuscrits sur la base de données MNIST.

Ces résultats ouvrent de nouvelles perspectives pour l'intégration du calcul fractionnaire dans les algorithmes d'apprentissage profond, offrant ainsi des améliorations potentielles pour diverses tâches de reconnaissance de formes et d'intelligence artificielle.

Références

[1] LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11), 2278-2324.

[2] Benahmed, N. (2002). Optimisation de réseaux de neurones pour la reconnaissance des chiffres manuscrits isolés, sélection et pondération des primitives par algorithmes génétiques. Thèse de maîtrise en génie de la production automatisée, Montréal.

[3] Belaid, A. (1995). *Reconnaissance des formes : Méthodes et applications*.

[4] Abdelhakim. D, (2011), « La reconnaissance des chiffres manuscrits par les machines à vecteurs de support(SVMs)»; Thèse de Master, Université de Tébessa.

[5] Shubair, A., et al. (2008). Off-line Arabic handwritten word segmentation using rotational invariant segments features. *The International Arab Journal of Information Technology*, 5(2), 125-131.

[6] Steinherz, T., Rivlin, E., & Intrator, N. (1999). «Off-line cursive word recognition: a survey». *International Journal on Document Analysis and Recognition*, 2(2), 90-110.

[7] Pal, U., & Jayadevan, R. (2011). Segmentation of handwritten text and recognition of segmented characters. In *ICDAR* (pp. 182-186).

[8] Chergui, L. (2012). *Combinaison de classifieurs pour la reconnaissance de mots arabes manuscrits*. Thèse de doctorat, Université de Mentouri, pp. 24-25.

[9] Trier, Ø. D., Jain, A. K., & Taxt, T. (1995). Feature extraction methods for character recognition—a survey. *Pattern Recognition*, 28(6), 739-766.

[10] Tremblay, G. (2004). *Optimisation d'ensemble de classifieurs non paramétriques avec apprentissage par représentation partielle de l'information*. Thèse de doctorat, École de technologie supérieure, Université du Québec.

[11] Thanh, T. H. (n.d.). *Le choix de paramètres pour la reconnaissance des chiffres manuscrits*. Diplôme de Magistère, University of Natural Sciences, HCMC, Vietnam.

[12] Soua, M. (2016). *Extraction hybride et description structurelle de caractères pour une reconnaissance efficace de texte dans les documents hétérogènes scannés: Méthodes et Algorithmes parallèles*. Université Paris-Est, pp. 37-38.

[13] Chevalier, S., et al. (2005). Étude de primitives spectrales pour la reconnaissance de caractères manuscrits dans le cadre d'une approche markovienne 2D. *DGA/Centre d'Expertise Parisien*, France, novembre 2005.

[14] Gonzalez, R. C., & Woods, R. E. (2008). *Digital image processing* (3rd ed.). Prentice Hall.

[15] Belaid, A., & Brlaid, Y. (1992). *Reconnaissance des formes : Méthodes et applications*, pp. 14-17.

[16] Ensiwiki, R. (2016). *Projet de reconnaissance hors-ligne des images dans les documents*. Université de Paris.

[17] Benamara, N., & Belaid, A. (1996). « Une méthode stochastique pour la reconnaissance de l'écriture arabe imprimée ». *Forum de la recherche en informatique*, Tunis, Tunisie.

[18] El-Yacoubi, A., Sabourin, R., Gilloux, M., & Suen, C.Y. (1998). Improved model architecture and training phase in an off-line HMM-based word recognition system. In *Proc. of the 13th International Conference on Pattern Recognition*, Brisbane, Australia, pp. 1521–1525.

[19] Benamara, N. (1999). *« Utilisation des modèles de Markov cachés planaires en reconnaissance de l'écriture arabe imprimée »*. Thèse de doctorat, Université des Sciences, des Techniques et de Médecine de Tunis II, Tunisie.

[20] Abdeldjalil, G. (2011), «Segmentation automatique pour la reconnaissance numérique des chèques bancaires Algériens »; Thèse de MAGISTER, centre Universitaire de Khanchela.

[21] Li, X., Hou, X., Li, H., Zhang, L., & Zhang, H. (2021). Deep Texture Analysis with Multi-Scale Features for Image Classification. *Sensors*, 21(6), 2136.

[22] Hochuli, A. G., Oliveira, L. S., Alceu, & Sabourin, R. (2018). Segmentation-Free Approaches for Handwritten Numeral String Recognition. arXiv:1804.09279v3 [cs.CV].

[23] Touzet, C. (1992). Les réseaux de neurones artificiels, Introduction au connexionnisme. Marseille, Juillet.

[24] Hinton, G. E. (2007). Boltzmann Machines. March 25.

[25] Glorot, Xavier and Bengio, Yoshua."Understanding the difficulty of training deep feedforward neural networks". In Proceedings of the International Conference on Artificial Intelligence and Statistics (AISTATS'10). Society for Artificial Intelligence and Statistics. 2010.

[26] Pierre-Luc. (2019). Dérivation fonction sigmoïde. Retrieved from https://urlz.fr/9EQz, January 15.

[27] Simon, F. (2019). Deep Learning, les fonctions d'activation. Retrieved from https://urlz.fr/9EQw, January 15.

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

[29] Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986). Learning representations by back-propagating errors. *Nature, 323*(6088), 533-536.

[30] Browne, M., Ghidary, S. S., & Mayer, N. M. (2007). Convolutional Neural Networks for Image Processing with Applications in Mobile Robotics. *December 2007*.

[31] Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986). Learning internal representations by error propagation. In Rumelhart, D. E., & McClelland, J. L. (Eds.), *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, 1,* 318-362. MIT Press, Cambridge, MA.

[32] LeCun, Y., Hinton, G. E. (1990). Dimensionality reduction and prior knowledge in e-set recognition. In Touretzky, D. S. (Ed.), *Advances in Neural Information Processing Systems,* 178-185. Morgan Kaufmann, San Mateo, CA.

[33] Touzet, C. (n.d.). *Les Réseaux de Neurones Artificiels: Introduction au Connexionnisme*. Retrieved from http://www.touzet.org/Claude/Web-Fac-Claude/Les_reseaux_de_neurones_artificiels.pdf.

[34] Indolia, S., Goswami, A. K., Mishra, S., & Asopa, P. (2018). Conceptual understanding of convolutional neural network - a deep learning approach. *Procedia Computer Science, 132,* 679-688.

[35] Hijazi, S., Kumar, R., & Rowen, C. Using Convolutional Neural Networks for Image Recognition. *Annals of Mathematical Statistics*, Cadence.

[36] Podlubny, I. (1998). *Fractional Differential Equations*. Academic Press.

[37] Abramowitz, M., & Stegun, I. A. (Eds.). (1972). *Handbook of Mathematical Functions with Formulas, Graphs, and Mathematical Tables*. Dover Publications.

[38] Kilbas, A. A., Srivastava, H. M., & Trujillo, J. J. (2006). *Theory and Applications of Fractional Differential Equations*. Elsevier.

[39] Mittag-Leffler, G. (1905). Sur la représentation analytique d'une branche uniforme d'une fonction monogène: Cinquième note. *Acta Mathematica, 29,* 101–181.

[40] Podlubny, I. (1999). *Fractional Differential Equations*. Academic Press.

[41] Wiman, A. (1905). Über den fundamentalsatz in der teorie der funktionen ea(x). Acta Mathematica, 29(1), 191–201.

[42] Haykin, S. S. (2009). *Neural Networks and Learning Machines* (3rd ed.). Pearson Education: Upper Saddle River, NJ, USA.

[43] Ruder, S. (2016). An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*.

[44] Duchi, J., Hazan, E., & Singer, Y. (2011). Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research, 12,* 2121–2159.

[45] Zeiler, M. D. (2012). ADADELTA: An adaptive learning rate method. *arXiv preprint arXiv:1212.5701*.

[46] Tieleman, T., & Hinton, G. (2012). *Neural Networks for Machine Learning*. Technical Report, Coursera.: Mountain View, CA, USA.

[47] Colab. (n.d.).

[48] Keras. (2024, May).

[49] Deng, L. (2012). The MNIST database of handwritten digit images for machine learning research (best of the web). *IEEE Signal Processing Magazine, 29*(6), p.141–142.

[50] Platt, J. C. (1999). Using analytic QP and sparseness to speed training of support vector machines. In *Advances in Neural Information Processing Systems* (pp. 557–563).

[51] Herrera-Alcántara, O. (2022). Fractional derivative gradient-based optimizers for neural networks and human activity recognition. *Applied Sciences*.