REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique
Université 8 Mai 1945 – Guelma
Faculté des Sciences et de la Technologie
Département de Génie Electrotechnique et Automatique

MEMOIRE

Présenté pour l'obtention du **diplôme** de **MASTER Académique**

Domaine : Sciences Et Technologie

Filière : Automatique

Spécialité : Automatique et Informatique industrielle

Par: BOUDJEHEM Mohammed Fakhr EL Islam
MOUMENI Younes

Thème

Impact Du Choix Des Fonctions D'activation Sur Les Performances De Divers Modèles CNN Pour Classifications D'images

Soutenu publiquement, le 22/06/2025, devant le jury composé de :

Babouri Abdesselam	Professeur	Univ. Guelma	Président
Debeche Mehdi	MAA	Univ. Guelma	Encadreur
El Aggoune Hocine	MCB	Univ. Guelma	Examinateur

Année Universitaire: 2024/2025

Dédicace:

À mon père, le meilleur père au monde, je tiens à exprimer ma profonde gratitude pour son encouragement constant, sa confiance, son soutien moral et matériel, et surtout pour son amour infini.

À ma mère, je dédie tout mon amour. Merci pour tous les sacrifices que tu fais chaque jour, pour la confiance que tu me portes et pour l'amour inconditionnel qui m'entoure.

À toute ma famille,

À mes chers amis,

À toutes les personnes qui occupent une place spéciale dans mon cœur,

Nous dédions également ce modeste travail.

En premier lieu, à ceux qui ne peuvent jamais être remplacés : nos parents, pour leurs sacrifices inestimables afin d'assurer notre éducation et notre bien-être.

Grâce à eux, nous avons pu suivre notre parcours de formation, qui constitue la base même de ce travail.

Que Dieu les bénisse et les protège.

À notre chère qui nous a soutenus dans les moments les plus difficiles

Remerciement :

Nous remercions Dieu Tout-Puissant de nous avoir accordé la santé, la force et la volonté nécessaires pour commencer et achever ce mémoire.

Nous exprimons notre profonde gratitude à notre encadrant, Monsieur Debeche Mehdi, pour son aide précieuse, la qualité exceptionnelle de son encadrement, sa patience, sa rigueur, ainsi que sa disponibilité tout au long de la préparation de ce travail.

Nous le remercions également pour son accompagnement pratique, son soutien moral et ses encouragements constants.

Nos remerciements s’adressent aussi à l’ensemble de nos enseignants, pour leur générosité, leur disponibilité et leur grande patience, malgré leurs nombreuses responsabilités académiques et professionnelles.

Résumé :

Dans le cadre de cette étude, nous explorons l'influence des différentes fonctions d'activation sur les performances des réseaux de neurones convolutionnels (CNN) appliqués à la classification d'images. Les CNN sont largement utilisés en vision par ordinateur en raison de leur capacité à extraire automatiquement des caractéristiques hiérarchiques à partir des données d'images. Cependant, le choix de la fonction d'activation peut avoir un impact significatif sur l'efficacité du modèle.

Nous avons évalué les performances de l'un des modèles CNN populaire, nomé ResNet18, en utilisant différentes fonctions d'activation telles que ReLU, Sigmoid, Tanh, et Swish, etc…. Les expérimentations ont été menées sur des ensembles de données d'images variées, incluant **MNIST, Fashion-MNIST,** et **SVHN (Street View House Numbers) Dataset**, pour garantir une évaluation complète.

Les résultats montrent que certaines fonctions d'activation, comme Swish, LeakyReLU, Mish, ELU, Softplus, Gelu, offrent des performances supérieures en termes de précision de classification, en particulier pour les modèles plus profonds et complexes. En revanche, les fonctions d'activation telles que Tanh, ELU, Softsign ont tendance à causer des problèmes de gradient vanishing, ce qui peut limiter l'apprentissage des modèles profonds.

Cette étude met en évidence l'importance du choix approprié des fonctions d'activation pour optimiser les performances des CNN dans les tâches de classification d'images. Elle fournit également des recommandations pratiques pour les chercheurs et les ingénieurs dans le développement de modèles CNN efficaces.

ملخص

في إطار هذه الدراسة، نستكشف تأثير دوال التفعيل المختلفة على أداء الشبكات العصبية الالتفافية (CNN) المطبقة على تصنيف الصور.

تُستخدم شبكات CNN على نطاق واسع في مجال الرؤية بالحاسوب، نظرًا لقدرتها على استخراج الخصائص الهرمية من بيانات الصور بشكل تلقائي. ومع ذلك، فإن اختيار دالة التفعيل قد يكون له تأثير كبير على كفاءة النموذج.

قمنا بتقييم أداء أحد أشهر نماذج CNN وهو ResNet18 باستخدام دوال تفعيل مختلفة مثلReLU ، Sigmoid، Tanh، و Swishوغيرها .وقد أُجريت التجارب على مجموعات بيانات متنوعة مثل MNIST و Fashion-MNISTو SVHN (Street View House Numbers)لضمان تقييم شامل.

أظهرت النتائج أن بعض دوال التفعيل مثلSwish :، LeakyReLU، Mish، ELU، Softplus، Gelu تُحقق أداءً أعلى من حيث دقة التصنيف، خاصةً مع النماذج العميقة والمعقدة.أما دوال مثلTanh :، ELU، Softsign فقد تؤدي إلى مشاكل "اختفاء التدرج (Vanishing Gradient) مما يُقيد تعلم النماذج العميقة

تُبرز هذه الدراسة أهمية الاختيار المناسب لدالة التفعيل لتحسين أداء شبكات CNN في مهام تصنيف الصور، كما تقدم توصيات عملية للباحثين والمهندسين في تطوير نماذج فعالة من.CNN

Sommaire

Liste des figures :

Chapitre 1 : Intelligence Artificielle et Réseaux de Neurones

Chapitre 2 : Les réseaux de neurones de convolution

Chapitre 3 : Les differentes fonctions d'activation des réseaux neurones et leur impact sur les performances de divers modèles CNN pour la classification

Chapitre 4 : L'implémentation sur MATLAB

Liste des Tableaux

Liste des acronymes utilisés dans le mémoire

OA: Over All Accuracy

AA: Average Accuracy

ADALINE : Adaptive Linear Neuron

ADAM : Adaptive Moment Estimation (optimizer)

CAH : Classification Ascendante Hiérarchique

CCE: Categorical Cross Entropy

CNN: Convolutional Neural Network

ELU: Exponential Linear Unit

FC: Fully Connected (layer)

GELU : Gaussian Error Linear Unit

IA : Intelligence Artificielle

MLP: Multi-Layer Perceptron

MNIST: Modified National Institute of Standards and Technology (dataset)

RELU: Rectified Linear Unit

SVHN: Street View House Numbers (dataset)

SVM : Support Vector Machine

Introduction générale

La classification d'images représente un enjeu crucial en vision par ordinateur, offrant des solutions avancées dans divers domaines tels que la reconnaissance d'objets, la conduite autonome, et l'analyse d'images médicales. Parmi les architectures de réseaux de neurones les plus performantes pour la classification d'images, les réseaux de neurones convolutionnels (CNN) se démarquent par leur capacité à capturer et à apprendre des caractéristiques spatiales complexes des données visuelles. Cependant, la performance de ces modèles est fortement influencée par plusieurs paramètres, parmi lesquels le choix des fonctions d'activation occupe une place prépondérante.

Les fonctions d'activation sont des composants essentiels des réseaux de neurones, déterminant la sortie des neurones en réponse à un ensemble d'entrées. Elles introduisent des non-linéarités qui permettent aux réseaux de neurones d'apprendre et de modéliser des relations complexes. Parmi les fonctions d'activation les plus couramment utilisées, on trouve la fonction ReLU (Rectified Linear Unit), connue pour sa simplicité et son efficacité, ainsi que les fonctions Sigmoid et Tanh, qui bien que puissantes, souffrent souvent de problèmes de gradient vanishing dans les réseaux profonds. Plus récemment, des fonctions telles que Swish et Mish ont été introduites, promettant d'améliorer encore davantage les performances des modèles CNN.

Cette étude vise à évaluer de manière systématique l'impact des différentes fonctions d'activation sur les performances de divers modèles CNN dans des tâches de classification d'images. Nous analyserons notamment des modèles populaires tels que ResNet18 en appliquant des fonctions d'activation variées. Les expériences seront menées sur des ensembles de données bien établis comme **MNIST, The Street View House Numbers (SVHN) Dataset**, **Fashion-MNIST**, offrant un cadre rigoureux pour comparer les résultats.

Les objectifs principaux de cette recherche sont :

1. Comparer les performances des modèles CNN avec différentes fonctions d'activation en termes de précision, vitesse de convergence, et robustesse.
2. Identifier les fonctions d'activation optimales pour des configurations de modèles spécifiques.
3. Fournir des recommandations pratiques aux chercheurs et praticiens pour le choix des fonctions d'activation dans les applications de classification d'images.

En conclusion, cette étude ambitionne de contribuer à l'amélioration des performances des modèles CNN en offrant une analyse approfondie et des lignes directrices claires sur le choix des fonctions d'activation, renforçant ainsi l'efficacité des solutions de classification d'images dans divers domaines d'application.

CHAPITRE 1 : INTELLIGENCE ARTIFICIELLE ET RESEAUX DE NEURONES.

CHAPITRE 1 : INTELLIGENCE ARTIFICIELLE ET RESEAUX DE NEURONES

1.1 Introduction

L'intelligence artificielle (IA) est aujourd'hui l'un des domaines les plus passionnants et influents de l'informatique moderne. Son but est de donner aux machines des capacités cognitives semblables à celles des humains, comme la reconnaissance vocale, la vision par ordinateur ou la prise de décision. Ce chapitre vous invite à plonger dans l'univers de l'IA en explorant ses définitions, son évolution au fil du temps, ses principales applications, ainsi que ses avantages, inconvénients et limites. Une attention particulière est accordée au deep learning et aux réseaux de neurones artificiels, qui sont au cœur des techniques d'apprentissage automatique d'aujourd'hui.

1.2 L'Intelligence Artificielle

1.2.1 Définition

L'intelligence artificielle (IA) est une branche de l'informatique qui consiste à créer des systèmes capables d'effectuer des tâches nécessitant normalement l'intelligence humaine, comme la reconnaissance vocale, la vision par ordinateur ou la prise de décisions. Elle s'appuie notamment sur la machine learning et le deep learning, qui permettent aux machines d'apprendre à partir de données, principalement grâce à des réseaux de neurones pour analyser des informations complexes comme des images ou des textes.**[1]**

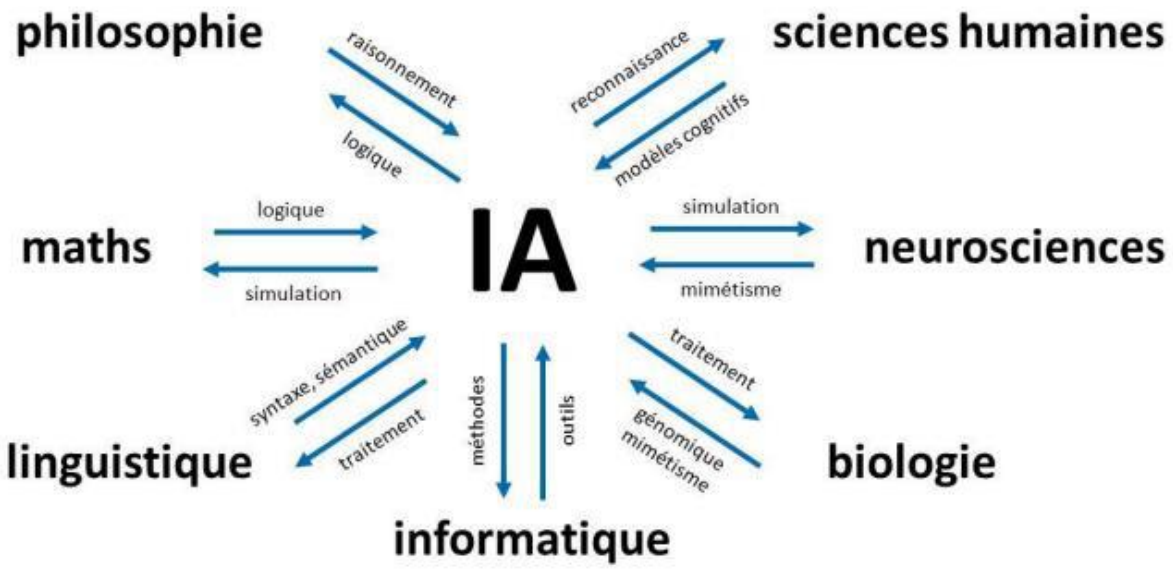


Figure 1.1 : Les types de l'intelligence artificielles**.**

1.2.2 Historique de l'intelligence artificielle

1950 :

Alan Turing a publié un article intitulé "Computing Machinery and Intelligence" dans lequel il a présenté un test appelé "Imitation Game", qui est maintenant connu sous le nom de test de Turing. Ce

test consiste à soumettre un candidat, humain ou machine, à une série de questions, et un juge doit déterminer si les réponses proviennent d'un être humain ou d'une machine. Aujourd'hui, le test de Turing est un concours où des participants tentent de convaincre le juge qu'ils sont des humains plutôt que des machines, en utilisant leur capacité à communiquer et à répondre de manière convaincante. **[2]**

1974-1980 :

Entre 1974 et 1980, l'IA a traversé ce qui est connu sous le nom de "l'hiver de l'IA". Ce terme fait référence à une période où le financement gouvernemental et l'intérêt pour le domaine de l'IA ont considérablement diminué **[2]**

1980 :

En août 1980, l'Université de Stanford a accueilli la première conférence nationale de l'American Society for Artificial Intelligence. **[2]**

1997 :

En 1997, le champion du monde d'échecs Garry Kasparov a été vaincu par le système IBM Deep Blue **[2]**

2002 :

En 2002, l'IA est entrée dans la maison pour la première fois sous la forme d'un aspirateur Roomba.

2011 :

Watson d'IBM a triomphé dans Jeopardy, où il a dû faire face à des questions et des énigmes complexes, et donner des réponses exactes

2014 :

Eugene Goostman a remporté un concours dans le célèbre "test de Turing", qui suscite de nombreuses controverses. Bien que l'on débatte de la réussite réelle d'Eugene au test, les rumeurs ont réussi à convaincre 33% des juges du panel qu'il était un véritable être humain. Ce résultat est survenu lors du 60e anniversaire de la mort d'Alan Turing, qui avait prédit qu'à l'approche de l'an 2000, les ordinateurs seraient suffisamment intelligents pour tromper les humains en les faisant croire qu'ils sont réels environ 30% du temps.

2017 *:*

Chaque jour, des avancées significatives dans le domaine de l'intelligence artificielle se produisent, et cette chronologie ne fait qu'effleurer la surface de ce vaste domaine. Elle constitue un point de départ pour un livre blanc qui explorera plus en détail l'impact de l'IA sur le monde des arts à but non lucratif et pourquoi il est important de s'en préoccuper. Restez à l'écoute pour découvrir comment l'IA vous permettra d'améliorer votre productivité et de travailler de manière plus intelligente. **[2]**

1.2.3 L'utilisation de l'intelligence artificielle

Ces dernières années, l'intelligence artificielle a trouvé sa place dans notre vie quotidienne et continue de progresser à grands pas. Voici quelques exemples d'utilisation courante de l'intelligence artificielle **[3][4]** :

- Les systèmes de messagerie électronique.
- Les réseaux sociaux.
- Les moteurs de recherche.
- Les outils de traduction.
- Les plateformes de commerce en ligne.
- Les applications de navigation.
- Reconnaissance d'image.
- Le calcul formel.
- La vision : analyse de texte, d'images.
- La robotique : génération de plans.
- Les machines autonomes : perception, interprétation, décision, action.
- Le langage naturel : traduction, compréhension, synthèse.
- La démonstration de théorèmes.
- Les jeux.
- La représentation des connaissances dans diverses disciplines (droit, médecine, automatique, etc.).

1.2.4 Les avantages, inconvénients et les limites de l'intelligence artificielle

- **Les avantages**

- **Augmentation de la productivité** : Automatisation des tâches répétitives, ce qui diminue les erreurs humaines et les coûts, tout en permettant de consacrer plus de temps aux activités créatives.**[4]**
- **Soutien à la prise de décision** : Analyse approfondie de données pour fournir des prévisions et des recommandations précises dans des secteurs comme la finance, la santé ou le marketing.
- **Traitement efficace de grandes quantités de données** : Capacité à repérer des tendances et des liens invisibles à l'œil humain.
- **Progrès dans la santé et la recherche médicale** : Diagnostiques assistés, traitements personnalisés, et avancées dans la recherche grâce à l'analyse de vastes bases de données médicales.
- **Réalisation de missions dans des environnements dangereux** : Exécution de missions risquées (espace, catastrophes, substances toxiques) au lieu des humains.

- **Les inconvénients**

- **Perte d'emplois** : L'automatisation peut supprimer certains emplois, notamment ceux qui impliquent des tâches répétitives, ce qui peut aggraver les inégalités sociales. **[4]**
- **Utilisations abusives** : Risques de cyberattaques, manipulation de l'information (deepfakes, fausses actualités), et surveillance intensive.
- **Manque de clarté** : Les décisions prises par certains systèmes d'IA restent difficiles à comprendre, ce qui complique la responsabilisation.
- **Biais et discriminations** : L'IA peut reproduire ou renforcer des inégalités sociales déjà présentes dans les données d'apprentissage.
- **Dépendance excessive** : Risque de perdre l'esprit critique et la capacité à décider par soi-même en confiant trop de responsabilités à l'IA.

1.3 Réseaux de neurones

1.3.1 Définition

Les **réseaux de neurones** sont des architectures inspirées du cerveau humain, utilisées dans **l'apprentissage profond** pour exploiter des données. Ils permettent de **prédire** de nouvelles valeurs, que ce soit par **régression** (pour des valeurs continues) ou par **classification** (pour des valeurs discrètes).**[10]**

1.3.2 Historique

1890: William James introduces the concept of **associative memory**, which forms the foundation of the later **Hebb's law**.**[11]**

1943: McCulloch and Pitts create a model of the **binarized formal neuron** and show that it can perform logical operations.

1949: Donald Hebb proposes the **Hebbian learning rule**, explaining animal conditioning through changing neural connections.

1957: Frank Rosenblatt develops the **Perceptron**, the first neural network model used for **pattern recognition**.

1960: Bernard Widrow introduces **ADALINE**, a model similar to the Perceptron but with a new learning rule, foreshadowing **gradient backpropagation**.

1969: Minsky and Papert emphasize the **limitations of the Perceptron**, especially its inability to solve certain non-linear problems.

1972: Teuvo Kohonen expands the idea of **associative memories** for pattern recognition purposes.

1982: John Hopfield refreshes interest with his **theory of recurrent networks**.

1983: The **Boltzmann machine** attempts to address the shortcomings of the Perceptron but remains impractical due to **high computational costs**.

1985: The **gradient backpropagation** algorithm is rediscovered, enabling **multi-layer networks** to learn complex and non-linear functions effectively.

1.3.3 Domaines d'application des réseaux de neurones artificiels

Aujourd'hui, les réseaux de neurones artificiels ont de nombreuses applications dans des secteurs très variés **[11] :**

- **Traitement d'images** : reconnaissance de caractères et de signatures, compression d'images, reconnaissance de forme, cryptage, classification, etc.
- **Traitement du signal** : filtrage, classification, identification de source, traitement de la parole…etc.
- **Contrôle** : commande de processus, diagnostic, contrôle qualité, asservissement des robots, systèmes de guidage automatique des automobiles et des avions…etc.
- **Défense** : guidage des missiles, suivi de cible, reconnaissance du visage, radar, sonar, lidar, compression de données, suppression du bruit…etc.
- **Optimisation** : planification, allocation de ressource, gestion et finances, etc.
- **Simulation** : simulation du vol, simulation de boîte noire, prévision météorologique, recopie de modèle…etc.

1.3.4 Les éléments fondamentaux d'un réseau de neurones artificielle

Comprennent les composants suivants :

- **Le perceptron simple**

Il fonctionne comme un classificateur linéaire. Dans sa version la plus basique, un réseau de neurones simple (ou perceptron simple) est constitué de : (**voir Figure 1.2) [12].**

- 1 couche d'entrée.
- 1 seule couche de sortie.
- Pas de couches caches.

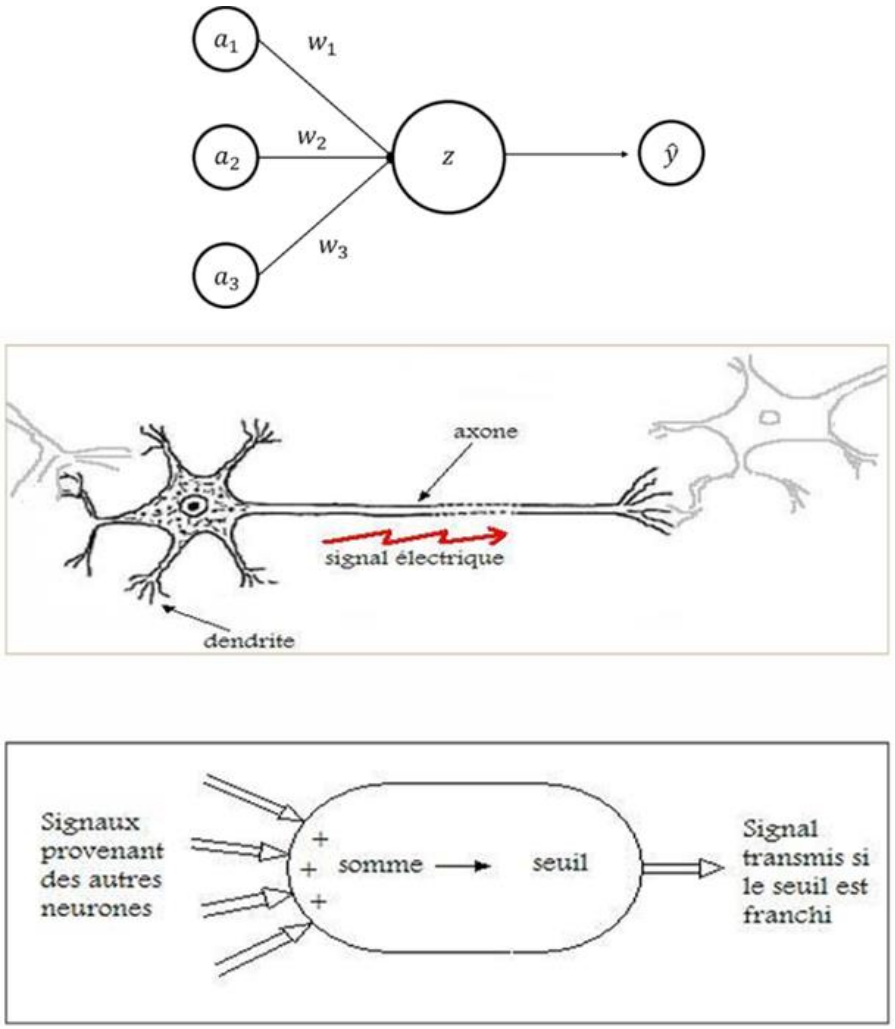


Figure 1.2 : Neurone de Modèle de perceptron.

- La sortie est une fonction du poids et des entrées :

$$\mathbf{z = \hat{y} = \left(\sum_{i=1}^{n=3} a_1 w_1 + a_2 w_2 + a_3 w_3 + b\right)} \text{.....(1.1)}$$

- Entrées: a_1, a_2, a_3
- Poids: w_1, w_2, w_3
- Biais: b
- Sortie: z

1.3.5 Le perceptron multicouche (MLP)

Dans le cadre du modèle de perceptron multicouche (**illustré à la Figure 1.3**), les perceptrons sont agencés en différentes couches. Ces perceptrons multicouches ont la capacité de traiter des données qui ne peuvent pas être séparées de manière linéaire. Avec l'émergence des algorithmes de rétropropagation, ils sont devenus le type de réseaux de neurones le plus couramment utilisé. Les

perceptrons multicouches (**MLP**) sont généralement structurés en trois niveaux : **la couche d'entrée**, la couche intermédiaire, souvent appelée **couche cachée**, et la **couche de sortie**. L'efficacité de l'utilisation de plusieurs couches cachées n'a pas encore été prouvée. Lorsque tous les neurones d'une couche sont reliés à ceux de la couche suivante, on désigne cela comme des couches entièrement connectées **[13].**

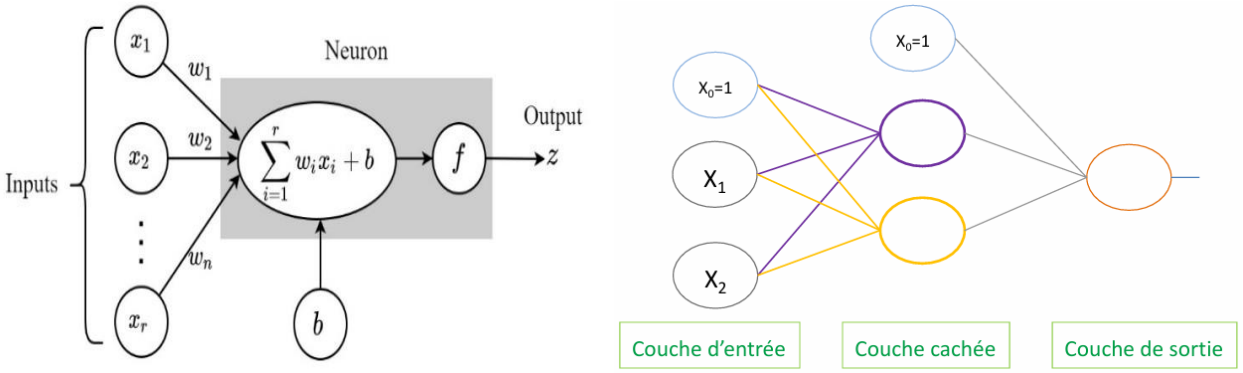


Figure 1.3: Neurone de modèle de perceptron multicouche (**MLP)**.

$$z = f * (\sum_{i=1}^{n}(w_i * x_i + b)\text{……..……..(1.2)}$$

f : fonction d'activation.

Par extension, on appelle couche d'entrée l'ensemble des **neurones d'entrée**, couche de sortie l'ensemble des **neurones de sortie**. Les couches intermédiaires n'ayant aucun contact avec l'extérieur sont appelées **couches cachées**.

La **figure 1.3** représente un réseau de neurones non bouclé qui a une structure particulière, très fréquemment utilisée : il comprend des entrées, deux couches de neurones cachés et un seul neurone de sortie. Les neurones de la couche cachée ne sont pas connectés entre eux. Cette structure est appelée Perceptron multicouches.

1.3.6 Fonctionnement des réseaux de neurones

Le principe des réseaux de neurones s'articule autour de trois étapes fondamentales :

- **Premièrement**

Pour chaque neurone d'une couche, il convient de multiplier la valeur d'entrée par le poids associé.

- **Deuxièmement**

Pour chaque couche, il est nécessaire de faire la somme de toutes les pondérations des neurones, à laquelle on ajoute un biais.

- **Troisièment**

La dernière étape consiste à appliquer une fonction d'activation sur cette somme afin d'obtenir une nouvelle sortie.

1.4 Conclusion

Ce chapitre aborde les réseaux de neurones et l'intelligence artificielle, ce dernier est un domaine en pleine croissance, avec des applications de plus en plus présentes dans notre vie quotidienne. Depuis ses débuts, marqués par des progrès notables et des périodes de stagnation, jusqu'à l'avènement des réseaux de neurones et du deep learning, l'IA offre des opportunités impressionnantes tout en soulevant des questions éthiques, sociales et techniques importantes. Comprendre son fonctionnement, notamment à travers les réseaux de neurones comme le perceptron et ses différentes versions (simple, multicouches), est important pour anticiper les défis futurs et exploiter au mieux ses potentialités.

CHAPITRE 2 : LES RESEAUX DE NEURONES DE CONVOLUTION.

CHAPITRE 2 : LES RESEAUX DE NEURONES DE CONVOLUTION.

2.1 Introduction

Le développement de l'apprentissage profond (deep learning) a changeé le domaine de l'intelligence artificielle, notamment dans la reconnaissance d'images, la vision par ordinateur et le traitement du langage naturel. Ce chapitre examine en détail les réseaux de neurones convolutifs (CNN), une architecture essentielle du deep learning. Après un bref rappel historique de leur évolution, avec des modèles tels que : ResNet18, LeNet et AlexNet…etc, il décrit leur structure, leur fonctionnement (convolution, pooling, ReLU, etc.), ainsi que leurs différentes variantes et usages. Le chapitre couvre aussi les méthodes de classification supervisée et non supervisée, les techniques pour optimiser l'apprentissage, les métriques pour évaluer leurs performances, ainsi que les algorithmes de rétropropagation utilisés lors de l'entraînement.

2.2 Histoire

Le premier réseau de neurones convolutif a été en réalité développé en 1998 par le chercheur français Yann LeCun. Ce réseau appelé LeNet a permis d'atteindre de très bonne performance en reconnaissance des caractères, Bien que cette approche donne des résultats, ses progrès et son évolution ont été limités par les progrès technologiques en matière de micro-processeurs, de puissance de calculs, et du manque d'accessibilités à des données afin de pouvoir entraîner les neurones. Cependant certains chercheurs ont continué à travailler sur ce modèle pendant environ deux décennies. Et, avec l'aide des évolutions en matière de technologies mais surtout avec la disponibilité toujours plus grande de données, ont pu améliorer cette technique. [18] Ce n'est qu'en 2012 que le Deep Learning est remis au goût du jour en remportant avec succès le concours de reconnaissance d'image fondé par l'université de Stamford (Large Squale Visual Recognition Challenge : ILSVRC) grâce à l'évolution en matière de technologies et à la disponibilité toujours plus grande de données. Un nouvel algorithme de Deep Learning explose les records Il s'agit d'un réseau de neurones convolutif appelé AlexNet, largement inspiré du réseau LeNet. ImageNet regroupant 15 000 000 d'images naturelles comportant différents objets et diverses scènes (véhicules, animaux,). Aujourd'hui, les réseaux de neurones convolutifs, sont toujours les modèles les plus performants pour la classification d'images. Google, Microsoft, Facebook, Baidu (le moteur de recherche chinois), Alibaba (site marchand chinois), Nvidia (géant du processeur graphique) …utilisent les CNN dans leurs applications. [19]

CHAPITRE 2 : LES RESEAUX DE NEURONES DE CONVOLUTION.

2.3 Apprentissage profond (deep learning)

2.3.1 Définition

Le **deep learning** (ou apprentissage profond) est une branche du **machine learning** qui emploie des **réseaux de neurones profonds**, composés de plusieurs couches cachées, pour analyser et comprendre des données complexes telles que la voix ou les images.

Les architectures d'apprentissage profond telles que les réseaux de neurones profonds, les réseaux neuronaux convolutifs « **convolutional deep neural networks** » **[5][6]**, ont plusieurs champs d'application :

- La vision par ordinateur (reconnaissance de formes).
- La reconnaissance automatique de la parole.
- Le traitement automatique du langage naturel.
- La reconnaissance audio.

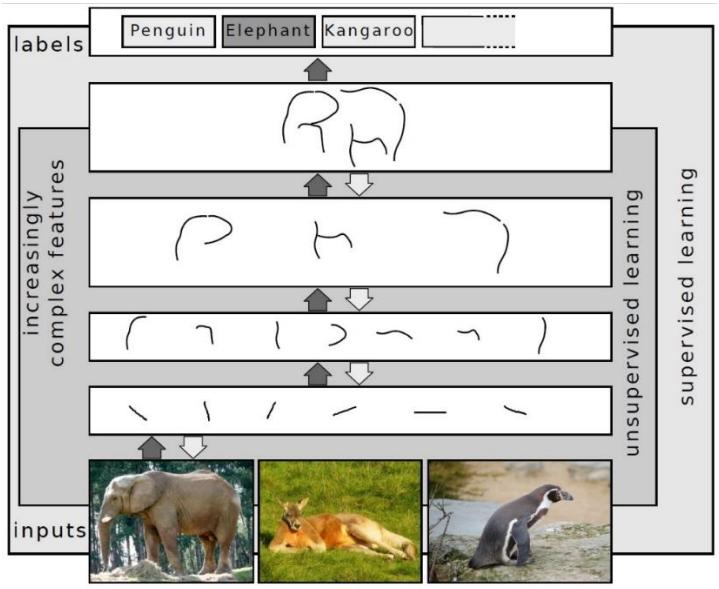


Figure 2.1: Plusieur couches d abstraction en appretissage profond

2.3.2 Historique

Le progrès du **deep learning** a connu une accélération importante dans les années 2010, grâce à la combinaison de quatre éléments essentiels **[7]**:

- La création de **réseaux de neurones à plusieurs couches**, issus du perceptron des années 1950,
- Des **algorithmes d'apprentissage** performants, inventés dans les années 1980,

- Une **augmentation de la puissance de calcul**
- Et l'accès à de **gigans bases de données**.

Ces progrès ont donné lieu à des résultats remarquables **[8]**:

- En **2015-2016**, **AlphaGo** (basé sur le deep learning) a vaincu les champions européens et mondiaux du jeu de go.
- En **2019**, **OpenAI** a présenté **GPT-2**, un générateur de texte très avancé, tout en soulignant les risques liés à un mauvais usage.

2.3.3 Domaines d'application d'apprentissage profond

L'apprentissage profond s'applique à divers secteurs des NTIC, notamment **[9]** :

- La reconnaissance visuelle par exemple, d'un panneau de signalisation par un robot ou une voiture autonome et vocale.
- La robotique
- La bioinformatique, p. ex., pour l'étude de l'ADN et des segments non codants du génome, ou encore la Cytométrie
- La reconnaissance ou la comparaison de forms
- La sécurité
- La santé
- La pédagogie assistée par l'informatique
- L'intelligence artificielle en général
- La traduction.

2.4 Réseaux de neurone convolutif

Actuellement, les modèles de réseaux de neurones convolutionnels sont les plus efficaces pour la classification d'images. Ils sont connus sous l'acronyme CNN, qui signifie Convolutional Neural Network en anglais, et se composent de deux éléments clairement définis (**Figure 2.1)**. Initialement, une image est présentée sous forme de matrice de pixels. Elle possède deux dimensions pour une image en échelle de gris. La couleur est symbolisée par une dimension supplémentaire, de profondeur 3, pour illustrer les couleurs primaires [Rouge, Vert, Bleu]. La partie convolutive, au sens strict du terme, constitue le premier segment d'un CNN. Elle opère en tant qu'extracteur de caractéristiques à partir des images. Une image subit un passage à travers une série de filtres, ou noyaux convolutifs, produisant de nouvelles images connues sous le nom de

cartes de convolution. Quelques filtres intermédiaires diminuent la résolution de l'image en procédant à une opération de maximum local. En fin de compte, les cartes et Les données issues des convolutions sont aplanies et combinées en un vecteur de caractéristiques, nommé code CNN. [20]

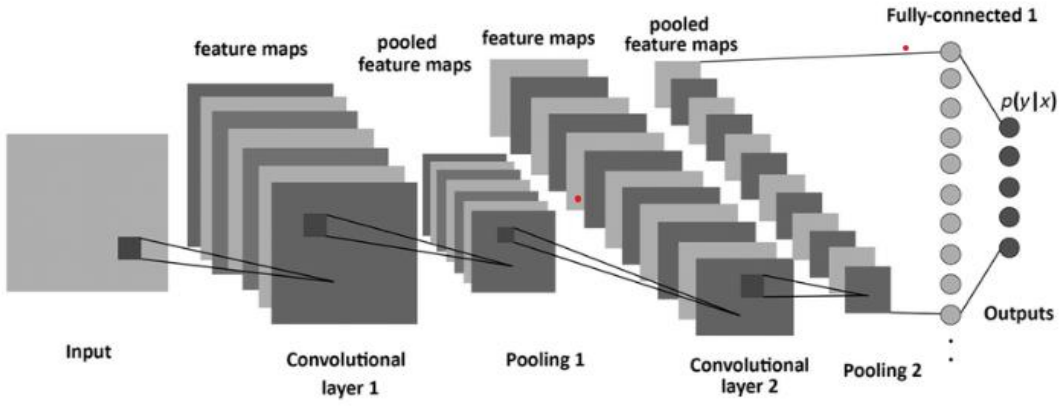


Figure 2.2: Architecture classique d'un réseau de neurones convolutifs

2.5 Architecture Les réseaux de neurones convolutionnels

Une architecture CNN est formée par un empilement de couches de traitement indépendantes :

- La couche de convolution (CONV) qui traite les données d'un champ récepteur.
- La couche de pooling (POOL), qui permet de compresser l'information en réduisant là
- La couche de correction (ReLU), souvent appelée par abus 'ReLU' en référence à la fonction d'activation (Unité de rectification linéaire).
- La couche "entièrement connectée" (FC), qui est une couche de type perceptron.
- La couche de perte (LOSS).

2.5.1 Couche de convolution (CONV)

Le CNN se compose de la couche de convolution qui en est l'élément fondamental. Le volume de la couche de convolution est déterminé par trois paramètres : **la profondeur, le pas** et **la marge**.

- **Profondeur de la couche :** quantité de noyaux de convolution (ou quantité de neurones liés àun champ identique de réception).
- **L'écart :** supervise la superposition des champs récepteurs. Plus, la taille du pas est réduite, plus les champs récepteurs se superposent et plus le volume de sortie sera important.
- **Marges (ou padding zéro)** : il est parfois pratique d'ajouter des zéros aux bords du volume d'entrée. Le troisième hyperparamètre est la dimension de ce 'zero-padding'. Cette marge

autorise la gestion de l'aspect spatial du volume généré en sortie. Il est parfois désiré de maintenir une surface identique à celle du volume d'entrée, en particulier.

Si la taille du pas et l'écart appliqué à l'image d'entrée servent à réguler le nombre de champs récepteurs En ce qui concerne la gestion de la surface de traitement, la profondeur offre un aperçu du volume de sortie. De manière similaire à une image qui peut posséder un volume, si l'on considère une profondeur de 3 pour les trois canaux RGB d'une image en couleur, la couche de convolution affichera également un volume en sortie. C'est pourquoi on évoque plutôt les termes de « volume sortant » et de « volume entrant ». Car une image ou la sortie d'une autre couche de convolution peut servir d'entrée à cette dernière.

On peut déterminer la dimension spatiale du volume de sortie en fonction de celle du volume d'entrée $\mathbf{W_i}$.

Le nombre de champs récepteurs qui définit la surface de traitement **K**, l'intervalle **S** est utilisé pour leur application, ainsi que la dimension de la marge L'expression utilisée pour déterminer le nombre de neurones dans le volume de sortie est :

$$\mathbf{W_0 = \frac{W_i - K + 2P}{S} + 1} \quad \textbf{(2.1)}$$

Si W_0 n'est pas un nombre entier, les neurones périphériques ne recevront pas autant d'entrées que les autres. Il sera donc nécessaire d'élargir la marge (pour générer des entrées virtuelles) Souvent, on considère un pas **S=1**, on calcule donc la marge de la manière suivante :

$$\mathbf{p = \frac{(w_0 - 1).s - w + k}{2}} \quad \textbf{(2.2)}$$

Si on souhaite un volume de sortie de même taille que le volume d'entrée. Dans ce cas particulier la couche est dite "connectée localement".

- **Principe du filtrage adaptatif**

Bien que l'utilisation de fenêtres de convolution soit devenue la norme dans les CNN, la façon dont nous pouvons démontrer que la convolution est une opération appropriée pour détecter des caractéristiques dans les signaux. Pour cela, nous nous inspirons de la théorie des filtres adaptatifs, grâce à laquelle la convolution du signal considéré. [14] Avec la caractéristique que nous recherchons confirmera la présence et la localisation de la caractéristique en question. Rappelons que la sortie du filtre adapté est calculée par convolution

L'équation mathématique du filtre de convolution dans un réseau de neurones convolutif (CNN) est donnée par :

$$\boldsymbol{y(n) = x(n) * w(-n) = \sum_{m}^{n} x(n+m)w(m) = x(n) * w(n)}\ldots\ldots\ldots\ldots\ldots\ldots \textbf{(2.3)}$$

- $y(n)$ Est la sortie du filtre de convolution pour le canal k.
- w(m) Représente les poids du filtre de convolution.
- x(n + m) Est l'entrée du signal ou de l'image.
- m Est la taille du filtre.

2.5.2 Couche de pooling (POOL)

Une autre notion clé des CNNs est le pooling, qui représente une méthode de réduction de l'échantillon de l'image. L'image d'entrée est divisée en une succession de rectangles non superposés de n pixels de côté (pooling). On peut considérer chaque rectangle comme une tuile. Le signal produit par la tuile est déterminé en fonction des valeurs que les divers pixels de la tuile peuventavoir

Le pooling diminue les dimensions spatiales d'une image intermédiaire, ce qui réduit le nombre de paramètres et le volume de calcul au sein du réseau. Dans une architecture CNN, il est courant de placer régulièrement une couche de pooling entre deux couches convolutives consécutives afin de maîtriserl'overfitting. Le processus de regroupement générait également une sorte d'invariance par translation.

La couche de pooling opère indépendamment sur chaque tranche de profondeur de l'entrée, se concentrant uniquement sur la mise à l'échelle au niveau de la surface. La méthode la plus fréquemment utilisée consiste en une couche de pooling avec des tuiles mesurant 2x2 (largeur/hauteur), et comme résultat, la valeur maximale obtenue en entrée. On fait référence ici à « Max-Pool 2x2 ».

Le pooling offre d'importantes améliorations en termes de puissance de calcul. Toutefois, à cause de la diminution radicale de la taille de la représentation (et par conséquent de la perte d'informations liée), l'approche privilégiée actuellement est l'emploi de petits filtres (par exemple, 2x2). On peut également omettre la couche de pooling, mais cela augmente le risque de surapprentissage.

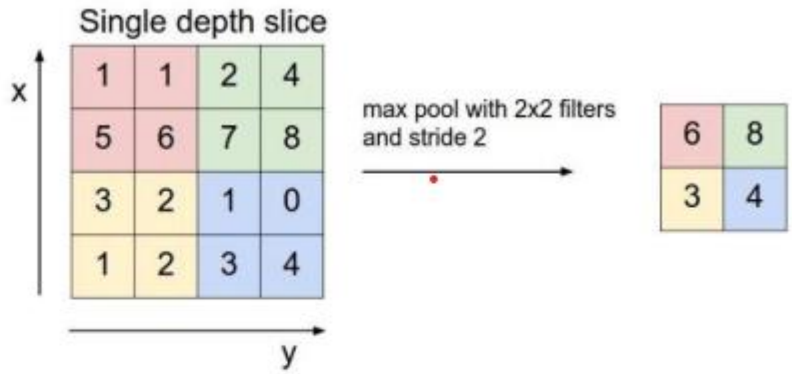


Figure 2.3: Exemple de fonctionnement de la couche pooling avec un filtre 2x2

Afin de réduire la taille potentiellement excessive du débit de données, les signaux de sortie à chaque couche sont généralement sous-échantillonnés davantage grâce à l'opération appelée "pooling", en complément du schéma de sous-échantillonnage par stride décrit précédemment. Une opération de pooling typique pour $\mathbf{y_k(n)}$ dans un réseau neuronal convolutif (CNN) est le max-pooling, qui divise le signal de sortie en segments non chevauchants de **P** échantillons et prend la valeur maximale de chaque segment. Pour une image, celle-ci est divisée en segments non chevauchants de P × P pixels, et la valeur maximale de chaque segment est sélectionnée. Le signal à la sortie de l'étape de max-pooling avec P segments devient ainsi.

$$\boldsymbol{O_k(n)} = \mathbf{max}\{\mathbf{f}(\boldsymbol{y_k(n)}), \mathbf{f}(\boldsymbol{y_k(n+1)}), \dots, \mathbf{f}(\boldsymbol{y_k(n+P-1)})\} \ldots\ldots\ldots\ldots \mathbf{.(2.4)}$$

où:

- $O_k(n)$ Est le signal de sortie après l'opération de pooling,
- $\mathrm{f}(y_k(n))$ Est la sortie activée du neurone convolutif kk,
- P Est la taille du segment de pooling (par exemple, **P × P** pour une image),
- La fonction max sélectionne la valeur maximale parmi les P échantillons.

Si l'on considère le average pooling, l'équation devient :

$$\mathbf{O_k(n)} = \frac{\mathbf{1}}{\mathbf{p}} \sum_{\mathbf{i=0}}^{\mathbf{p-1}} \mathbf{f}(\mathbf{y_k(n+i)}) \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots .. (\mathbf{2.5})$$

- **Influence du pooling sur l'apprentissage des CNN :**

- Réduction de la dimension : En diminuant le nombre de paramètres, le pooling prévient le surajustement et accélère l'entraînement.

- Stabilité face aux variations spatiales : En sélectionnant les valeurs maximales ou moyennes sur des zones locales, le réseau devient plus robuste aux déplacements des caractéristiques dans l'image.
- Moins de calculs : En réduisant la taille des activations intermédiaires, il diminue la charge de calcul et la consommation de mémoire.
- Propagation de l'information : Le pooling agit comme une opération de filtrage qui conserve uniquement les informations les plus pertinentes avant de les transmettre aux couches suivantes.

Il existe plusieurs types de pooling :

- **Max-pooling** : Sélectionne la valeur maximale dans une région donnée.
- **Average-pooling** : Prend la moyenne des valeurs de la région.
- **Global pooling** : Effectue une opération de pooling sur l'ensemble de la carte des caractéristiques, souvent utilisé pour condenser l'information avant la classification finale.

Le choix du type de pooling dépend du problème à résoudre. Par exemple, le max-pooling est souvent privilégié pour les tâches de reconnaissance d'objets, car il conserve les caractéristiques les plus saillantes.

2.5.3 Couches de correction (RELU):

On peut optimiser l'efficience du traitement en insérant entre les niveaux de traitement une strate qui exécutera une opération mathématique (fonction d'activation) sur les signaux sortants. La fonction ReLU, qui signifie Unités Linéaires Rectifiées $\mathbf{:F(x) = max(0, x)}$ Cette fonction force les neurones à retourner des valeurs positives.

2.5.4 Couche entièrement connectée (FC):

Après une succession de couches convolutives et de max-pooling, le processus de réflexion avancé au sein du réseau neuronal s'effectue à travers des couches entièrement reliées. Dans une couche entièrement connectée, chaque neurone est relié à toutes les sorties de la couche antérieure. On peut donc calculer leurs fonctions d'activation en utilisant une multiplication matricielle.

2.5.5 Couche de perte (LOSS):

La fonction de perte détermine comment l'apprentissage du réseau sanctionne la divergence entre le signal anticipé et celui effectivement produit. Elle est généralement la couche finale dans le

réseau. On peut utiliser différentes fonctions de perte, adaptées à diverses tâches, dans ce contexte.

La fonction « Softmax » sert à déterminer la distribution des probabilités des classes de sortie.

- **L'entropie croisée catégorielle (CCE) :**

Également connue sous le nom de perte softmax ou de perte logarithmique, est l'une des fonctions de perte les plus couramment utilisées dans l'apprentissage automatique, en particulier pour les problèmes de classification. Elle mesure la différence entre la distribution de probabilité prédite et la distribution réelle (vraie) des classes. Cette fonction aide un modèle d'apprentissage automatique à déterminer à quel point ses prédictions sont éloignées des véritables étiquettes et l'aide à apprendre à faire des prédictions plus précises.

- **Comprendre l'entropie croisée catégorielle :**

L'entropie croisée catégorielle est utilisée lorsque le problème de classification comporte plus de deux classes (classification multi-classes). Elle mesure la différence entre deux distributions de probabilités : la distribution de probabilités prédite et la distribution réelle, qui est représentée par un vecteur codé à un point. Dans un vecteur codé à un parmi n classe, la classe correcte est représentée par « 1 » et toutes les autres classes par « 0 ». L'entropie croisée catégorielle pénalise les prédictions en fonction du degré de confiance du modèle dans la classe correcte.

Si le modèle attribue une forte probabilité à la vraie classe, l'entropie croisée sera faible. Inversement, si le modèle attribue une faible probabilité à la classe correcte, l'entropie croisée sera élevée.

Représentation mathématique de l'entropie croisée catégorielle La formule de l'entropie croisée catégorielle s'exprime comme suit :

$$\mathbf{L}(\mathbf{y}, \hat{\mathbf{y}}) = -\sum_{\mathbf{i=1}}^{\mathbf{C}} \mathbf{y_i} \log(\hat{\mathbf{y}}_\mathbf{i}) \dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots (\mathbf{2.6})$$

Où :

$\mathbf{L}(\mathbf{y}, \hat{\mathbf{y}})$ Est la perte d'entropie croisée catégorielle.

$\mathbf{y}\_(\mathbf{i})$ Est la véritable étiquette (0 ou 1 pour chaque classe) du vecteur cible codé à un coup.

$\hat{\mathbf{y}}_\mathbf{i}$ Est la probabilité prédite pour la classe

$\mathbf{C}$ Est le nombre de classes

Dans cette formule, le logarithme garantit que les prédictions incorrectes sont fortement pénalisées.

2.6 Choix des hyperparamètres :

Les réseaux de neurones convolutifs emploient un plus grand nombre d'hyperparamètres que le perceptron multicouche classique. Bien que les règles standards concernant les taux d'apprentissage et les constantes de régularisation demeurent en vigueur, il est nécessaire d'intégrer des concepts tels que le nombre de filtres, leur configuration et la structure du max pooling.

2.6.1 Nombre de filtres:

Étant donné que la dimension des images intermédiaires se réduit à mesure qu'on progresse en profondeur, les couches situées près de l'entrée ont généralement moins de filtres, tandis que les couches se trouvant plus près de la sortie peuvent en contenir un plus grand nombre. Afin de maintenir une consistance dans le calcul à chaque niveau, on privilégie généralement un produit constant entre le nombre de caractéristiques et le volume de pixels manipulés à travers les différentes couches. Afin de garantir la conservation des données d'entrée, il est nécessaire que le nombre de sorties intermédiaires (qui correspond au produit du nombre d'images intermédiaires et du nombre de positions de pixel) augmente généralement d'une couche à l'autre. La force du système est directement liée au nombre d'images intermédiaires, qui dépend à la fois de la disponibilité des exemples et de la complexité du traitement.

2.6.2 Forme du filter:

Les types de filtres varient considérablement dans la bibliographie. Leur sélection est souvent basée sur le jeu de données utilisé. Les résultats les plus performants sur les images MNIST (28 x 28) sont généralement observés avec des filtres de 5 × 5 en première couche, tandis que pour les ensembles d'images naturelles (fréquemment composés de centaines de pixels par dimension), on privilégie des filtres plus larges en première couche, tels que 12 × 12 ou même 15 × 15. Il s'avère donc essentiel de déterminer le niveau approprié de granularité afin d'élaborer des abstractions qui correspondent à l'échelle adéquate pour chaque situation.

2.6.3 Forme du Max Pooling:

Les valeurs standard sont 2 × 2. Un pooling 4 × 4 pourrait être justifié par des volumes d’entrée très importants dans les premières couches. Toutefois, l'utilisation de formes plus grandes risque d'entraîner une réduction significative du volume du signal, ce qui pourrait nuire à la conservation de l'information.

CHAPITRE 2 : LES RESEAUX DE NEURONES DE CONVOLUTION.

2.7 Les modèles réseaux de neurones convolutifs :

Il existe plusieurs modèles dans le domaine des réseaux convolutifs qui ont un nom. Les plus courants sont:

2.7.1 LeNet-5:

LeNet est le réseau neuronal convolutif prototype créé par Yann LeCun en 1990 et perfectionné ultérieurement en 1998. LeNet, réputé pour son efficacité, a été employé pour l'analyse des codes postaux,deschiffres.

Cette structure comprend 4couches convolutives (CONV) et de regroupement (Pooling) qui se succèdent, suivies de 3 couches entièrement interconnectées. LeNet était la première architecture de réseau de neurones convolutif, qui a non seulement réduit le nombre de paramètres, mais a également réussi à apprendre automatiquement les caractéristiques des pixels bruts. **La figure (2.4)** [22]

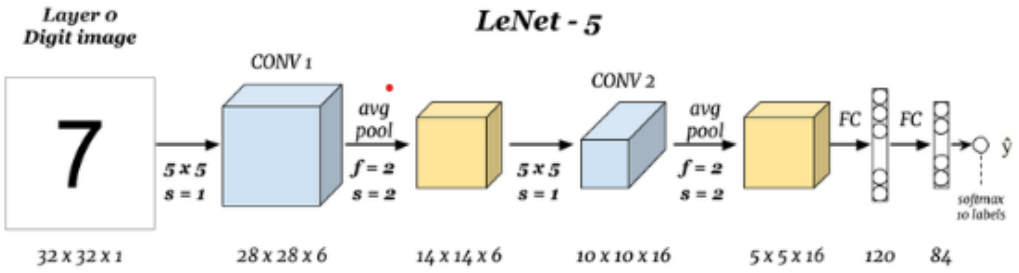


Figure 2.4: Architecture CNN LeNet-5

2.7.2 AlexNet:

AlexNet, qui a grandement contribué à populariser les réseaux de neurones convolutifs dans le domaine de la vision par ordinateur, est l'œuvre d'Alex Krizhevsky, Ilya Sutskever et Geoff Hinton. Cette architecture CNN renommée a été introduite au concours ImageNet ILSVRC en 2012 et a surpassé de manière significative les performances du deuxième candidat.

AlexNet se compose de 5 couches de convolution utilisant des unités linéaires rectifiées (ReLU) comme fonctions d'activation, accompagnées de 3 couches Max Pooling et 3 couches entièrement connectées (FC). La figure (2.5) [23]

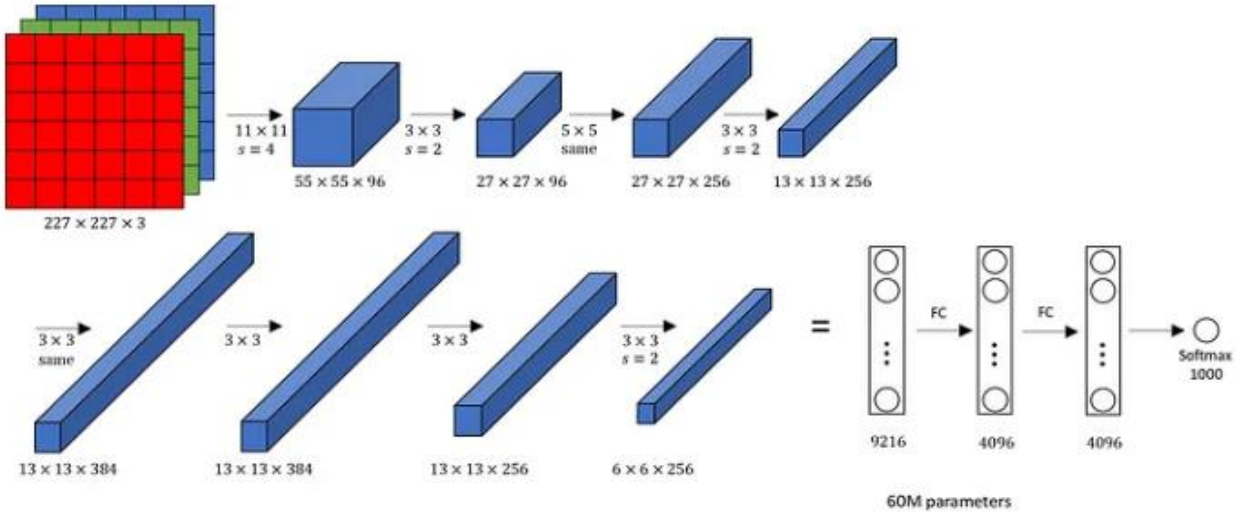


Figure 2.5: Architecture AlexNet

- **Structure du réseau AlexNet**

AlexNet a été proposé par Krijevski et Al., et a remporté le championnat dans la compétition 2012 Image Net ce qui plus que doublé la reconnaissance de précision de ImageNet. Le modèle se compose d'une structure de 8 couches et a sa possession exceptionnelle Advan-in image classification La structure spécifique d'AlexNet Le format d'entrée des données sources est de 227 227 3 pixs, par lequel 227 pix représente la largeur et la hauteur de l'entrée Im-âge 3 Représente les données sources est un mode RVB à trois canaux ainsi Prend en charge les photos en couleur dans les formats couramment utilisés donc il n'y a pas besoin de formats supplémentaires pour les sources de données en langue source recueillies Recadré. Le processus de calcul des deux premières couches est convolu- (Conv), ReLu, mutualisation maximale (max-pooling) et normaliza-(normal). Le résultat de sortie de la deuxième couche est 256 Fonction de carte pour opération de convolution dans laquelle la taille du noyau est 5, la foulée est 1, et les autres paramètres sont les mêmes que première couche. Les troisièmes et quatrièmes couches seulement effectué convolution et ReLu Operations. La cinquième couche est similaire à la première couche ex-CEPT qu'il n'a pas été Normalisé. Convertir le résultat de la cinquième couche dans un vecteur long et l'a entré dans un réseau neuronal traditionnel travail En utilisant une structure complètement connectée à trois couches. Les cœurs des deux premiers Calques complètement connectés sont respectivement 4.096. La dernière couche Génère 1000 nœuds et l'étiquette de valeur peut être obtenue en utilisant la régression fonctionnelle Softmax. [24]

2.7.3 GoogLeNet

GoogleNet la figure (2.6), également connu sous le nom d'Inception-V1, a remporté le concours ILSVRC de 2014. Il a été conçu par une équipe de Google dirigée par Christian Szegedy. Ce réseau de neurones convolutif se base sur les modules Inception et utilise des blocs qui encapsulent des filtres de tailles variées (1x1, 3x3 et 5x5) afin de capturer des informations à différentes échelles spatiales, avec l'emploi du filtre Concat pour la concaténation des résultats des filtres. De plus, la densité de connexion a été réduite grâce à l'utilisation d'une moyenne globale plutôt qu'une couche entièrement connectée. Ces ajustements ont conduit à une réduction significative du nombre de paramètres, passant de 60 millions à 4 millions [25]

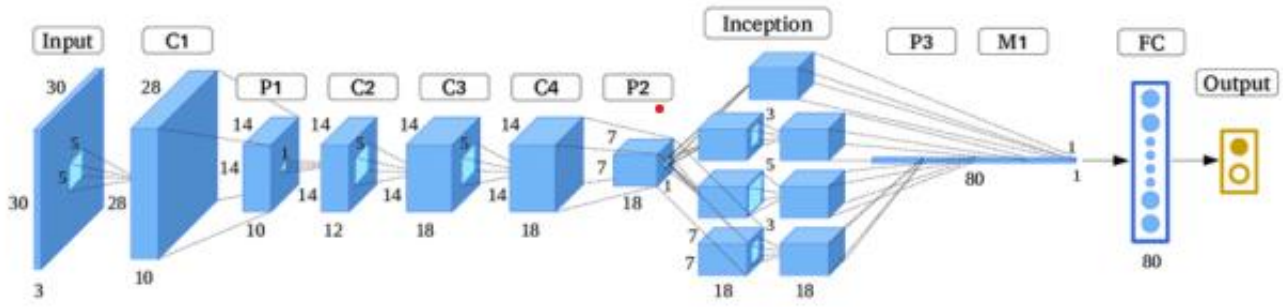


Figure 2.6: Module de démarrage de GoogLeNet

2.7.4 ResNet (2018)

Un réseau neuronal convolutif profond (CNN) composé de 18 couches, principalement composées de blocs résiduels. Ces blocs sont la marque de fabrique des ResNets, qui utilisent des connexions de saut pour faciliter l'apprentissage des réseaux profonds en sautant des couches. Dans sa forme originale, ResNet18 se compose de 17 couches convolutives, d'une couche entièrement connectée et d'une couche de classification SoftMax (la figure 2.7). **[26]**

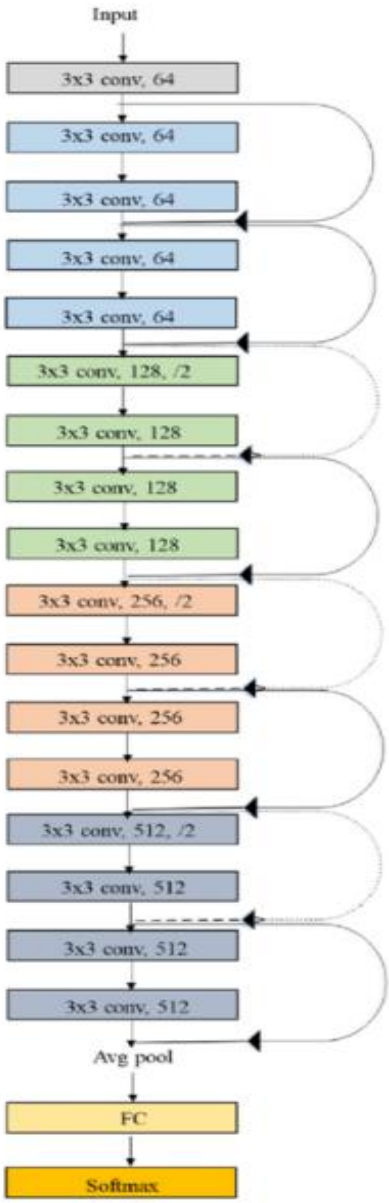


Figure 2.7: L'architecture ResNet 18

2.8 L'Applications des CNN

On utilise les CNN dans diverses applications, y compris :

• Classification d'Images : Reconnaître les éléments présents dans une image.

• Identification d'Objets : Repérer et catégoriser divers objets sur une image.

• Décomposition d'Images : Fractionner une image en segments qui correspondent à divers objets ou zones.

• Identification par biométrie faciale : Reconnaître et valider les traits du visage humain.
• Étude Vidéo :Dégager des données significatives à partir des séquences vidéo.
Les réseaux de neurones convolutifs ont transformé le domaine de la vision par ordinateur en raison de leur aptitude à acquérir automatiquement des caractéristiques sophistiquées et à optimiser considérablement la précision des modèles d'analyse d'images.

2.9 Caractéristiques et avantages

- L'un des atouts considérables des réseaux convolutifs réside dans l'emploi d'un poids commun pour les signaux qui pénètrent tous les neurones d'un même noyau de convolution.
- Cette technique diminue l'usage de la mémoire, optimise les performances et favorise une invariance du traitement par translation.
- C'est l'atout majeur du réseau de neurones convolutifs comparé au perceptron multicouche, qui traite chaque neurone de manière indépendante et attribue par conséquent un poids distinct à chaque signal d'entrée.
- Quand le volume d'entrée fluctue au fil du temps (vidéo ou audio), il serait pertinent d'ajouter un paramètre à travers l'échelle temporelle lors de la configuration des neurones. On désignera ceci comme un réseau neuronal à délai temporel (TDNN).
- Par rapport à d'autres méthodes de classification d'images, les réseaux de neurones convolutifs nécessitent un prétraitement relativement minimal.
- Cela implique que le réseau prend en charge de manière autonome l'évolution de ses filtres (apprentissage non supervisé), ce qui n'est pas forcément le cas pour d'autres algorithmes plus conventionnels.
- Un avantage crucial des CNN est l'absence de nécessité d'un paramétrage initial ou d'intervention humaine.

2.10 Méthodes de régularisation

En apprentissage automatique, la régularisation est un procédé visant à améliorer les performances de généralisation d'un algorithme d'apprentissage, autrement dit à diminuer son erreur sur échantillons de test. Cela peut éventuellement être réalisé au détriment de l'erreur d'apprentissage. Un tel procédé a pour but d'éviter le sur-apprentissage qui résulte d'une adaptation trop forte du modèle aux données d'entraînement. Du point de vue du compromise biais/variance, le sur-apprentissage décrit un modèle capable de très bien s'adapter à tout ensemble d'apprentissage

donné (faible biais) mais devrait fortement modifier ses paramètres (poids) pour s'adapter à un autre jeu de données d'apprentissage (forte variance). [27]

2.11 Méthodes de classification d'images

2.11.1 Définition de la classification

La classification est une tâche qui tente à identifier les objets en se basant sur certaines de leurs caractéristiques. Par exemple, pour pouvoir utiliser les images pour les analyses complémentaires ou pour la cartographie, il est souvent important de traduire l'information de fréquence contenue dans les images en information thématique portant sur l'occupation du sol ou la couverture végétale. On a généralement le choix entre deux approches : la classification supervisée et non supervisée

2.11.2 Méthodes supervisées

Dans le cas de l'apprentissage supervisé, on dispose d'un ensemble de données étiquetées, ou d'exemples qui se sont vus associés une classe par un professeur ou un expert. Cet ensemble d'exemples constitue la base d'apprentissage. Les méthodes d'apprentissage supervisé se donnent alors comme objectif général déconstruire à partir de la base d'apprentissage, ou fonctions de classement. Une telle fonction permet, à partir de la description d'un objet, de reconnaître un attribut particulier, la classe (Figure 2.8) [28]

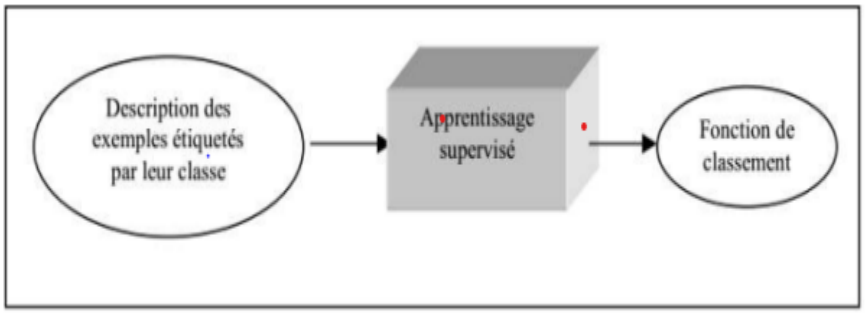


Figure 2.8: L'apprentissage supervise

L'inférence inductive est définie comme un processus qui à partir d'une connaissance spécifique observée sur certains objets et d'une hypothèse inductive initiale, permet d'obtenir une assertion inductive impliquant ou rendant compte fortement ou faiblement des observations. Dans le cas de l'apprentissage inductif supervisé, qui est un sous domaine de l'inférence inductive, la connaissance spécifique consiste en un ensemble d'objets appartenant à des classes connues.

CHAPITRE 2 : LES RESEAUX DE NEURONES DE CONVOLUTION.

L'assertion inductive est exprimée par une règle de classification qui assigne une classe à chaque objet. L'implication forte est satisfaite si la règle classe correctement tous les objets connus

Exemple

Nous allons illustrer le concept de la classification supervisé avec un exemple. Sois un enfant à qui on apprond à reconnaître des fruits. On lui donne des exemples avec leurs noms (**Tableau2.1)** :

Tableau2.1: Classification de fruits

Couleur	Taille	Fruit
Rouge	Petite	Cerise
Jaune	Grande	Banane
Rouge	Grande	Pomme
Jaune	Petite	Citron

Il observe ces exemples. Chaque fruit est décrit par des caractéristiques (couleur, taille) et a *une* étiquette (le nom du fruit). Plus tard, on te montre un nouveau fruit :

- Couleur : Rouge
- Taille : Petite

Il a rappelé que ce profil correspondait à une **Cerise** → Il dit : « C'est une cerise *! »*

Il a classé ce nouveau fruit en utilisant ce qu'Il a appris auparavant — c'est ça**,** la classification supervisée**.**

- **Algorithmes Méthodes supervisées**

Pour résoudre les taches de la classification supervisée, plusieurs algorithmes sont possibles, nous mentions les plus employées :

a) Les *K*- plus proches voisins (Kppv) :

Il s'agit de l'algorithme de classification le plus basique et le plus répandu, connu sous le nom « nearest neighbors (K-NN) » (ou « plus proches voisins » en français) la figure (2.9). Le principe fondamental de cette règle classification est que l'étiquette d'un point $x \in X$ est correctement déterminée par celle de ses voisins dans X. Une règle de classification est établie en déterminant que l'étiquette attribuée à un point $x \in X$ est celle qui prévaut parmi les $\boldsymbol{k}$ voisins les plus proches

dans l'échantillon. L'utilisateur peut librement déterminer le nombre ***k*** de voisins à prendre en compte ainsi que la distance sur *X* qui définit ce qu'est un « voisin dans *X* ». [29]

Pour bien illustrer cette définition on va se servir de l'exemple ci-dessous, un jeu de données avec deux classes bleu et rouge et on veut classer le nouveau point vert, L'algorithme consiste à prendre les plus proches voisins pour déterminer à quelle classe appartient ce point.

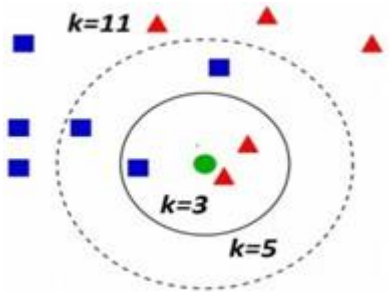


Figure 2.9: Schéma d'une classification par la méthode k plus proche voisin

Disons que **K = 3 (3-NN**), par conséquent le point appartient à la couleur rouge car c'est la classe majoritaire parmi ses **k** plus proche voisin.

Et si on prend **K = 5**, le point appartient à la couleur bleue. Enfin avec

K=11, notre point appartient à la couleur bleue

b) Machines à Vecteurs de Support (SVM)

L'objectif des SVM est de déterminer une séparation entre deux classes qui soit le plus éloignée possible de chaque point des données d'apprentissage. Si nous parvenons à identifier un séparateur linéaire, c'est-à-dire un hyperplan distinctif, le problème est considéré comme linéairement séparable. Dans le cas contraire, il n'est pas linéairement séparable et aucun hyperplan distinctif ne peut être établi. |30]

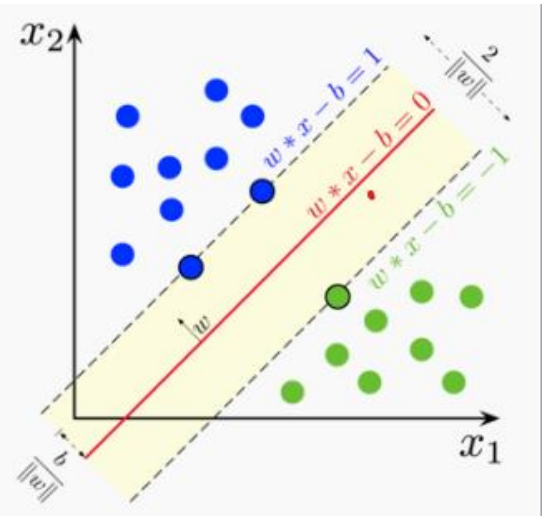


Figure 2.10: Machine à vecteurs de support

c) Réseau de Neurone

Les réseaux de neurones sont à l'origine d'une tentative de modélisation mathématique du Cerveau humain. Le principe général consiste à définir des unités simples appelées neurones, chacune étant capable de réaliser quelques calculs élémentaires sur des données numériques. On relie ensuite un nombre important de ces unités formant ainsi un outil de calcul puissant

Afin de mieux comprendre la technique de classification par les réseaux de neurones qui fonctionnent tous avec la même maniéré, nous utilisons l'algorithme de base des réseaux de neurones de la classification supervisé, qui est le perceptron, dédié à la classification binaire, c'est-à-dire séparation linéaire des données en deux classes, et nous réalisons un exemple très simple rendant le principe très clair. Le perceptron appelé neurone artificiel ou neurone formel, est inspiré de neurone biologique ou les entrés de neurone artificiel corresponde aux dendrites biologiques, et la sortie de neurone artificiel corresponde à l'axone de neurone biologique voir (figure2.11)

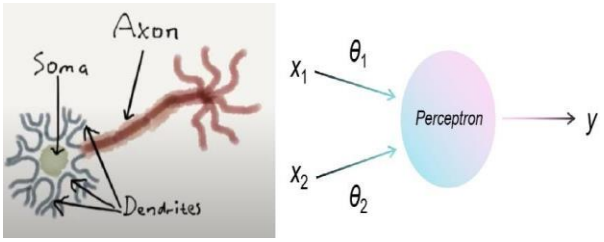


Figure2.11 : Neurone biologique et neurone artificiel

2.11.3 Méthodes non-supervisées:

L'apprentissage non-supervisé, encore appelé apprentissage à partir d'observations ou découverte, consiste à déterminer une classification « sensée » à partir d'un ensemble d'objets ou de situations données (des exemples non étiquetés). On dispose d'une masse de données indifférenciées, et l'on désire savoir si elles possèdent une quelconque structure de groupes. Il s'agit d'identifier une éventuelle tendance des données à être regroupées en classes. Ce type d'apprentissage, encore appelé Cluster ING ou Cluster Analysais, se trouve en classification automatique et en taxinomie numérique. Cette forme de classification existe depuis des temps immémoriaux. Elle concerne notamment les sciences de la nature (Figure 2.12), les classifications des documents et des livres mais également la classification des sciences élaborées au cours des siècles par les philosophes. [32]

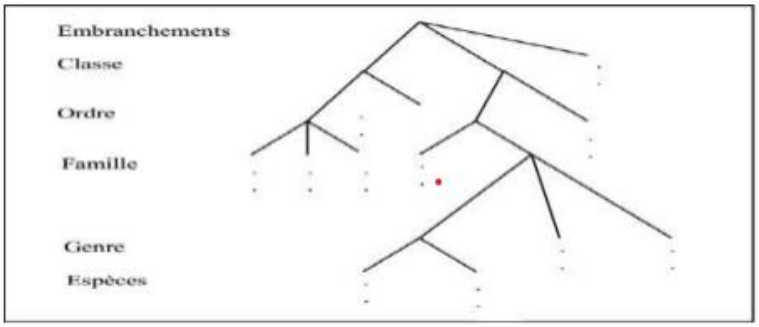


Figure 2.12: Extrait de la classification taxinomique de Linné

L'automatisation de la construction de classification constitue aujourd'hui un véritable domaine de recherche. La notion clé utilisée pour créer des classes d'objets est une mesure de la similarité entre les objets. Les classes ou concepts sont construits de façon à maximiser la similarité intraclasses et à minimiser la similarité interclasses. L'apprentissage non supervisé correspond également à la classification conceptuelle, où une collection d'objets forme une classe si cette classe peut être décrite par un concept, compte tenu d'un ensemble de concepts prédéfinis.

Exemple

Une boîte pleine de formes en papier. Il ne sait pas comment elles s'appellent, ni à quelle catégorie elles appartiennent. Il observe seulement leur forme et leur taille**.** Il doit regrouper ces objets par similarité, sans savoir leurs noms. (Tableau2.2)

Tableau2.2 : Des formes à classer sans étiquette

Forme	**Taille**
Carré	petite
Cercle	grande
Carré	grande
Triangle	petite

Groupe **1** : Carré petit, Carré grand → ce sont des carrés

Groupe **2** : Cercle petit, Cercle grand → ce sont des cercles

Groupe **3** : Triangle petit → forme différente

Il vient de faire de la classification non supervisée :

Sans connaître les étiquettes à l'avance**,** Il a regroupé les objets par resemblance

- **Algorithmes des méthodes non-supervisées**

On distingue une vaste gamme de techniques pour la classification non supervisée, qui se regroupent en deux catégories principales : les algorithmes de partition tels que : **K-moyennes**, **C-means floues**, **K-médoïdes**, etc., et les algorithmes hiérarchiques comme la Classification Ascendante Hiérarchique (CAH). Nous mentionnons.

a) L'algorithme du K-moyennes (k-means)

C'est une technique dont le but est de diviser des observations en **k** partitions dans lquelles chaque observation appartient à la partition avec la moyenne la plus proche. Nous citons deux méthodes connues sur le principe de k-means sont :

- Méthodes de centres mobiles
- Méthodes des nuées dynamiques.

b) Méthode de centres mobiles

Cette méthode consiste à construire une partition en k classes en sélectionnant k individus commence, des classes tirées au hasard de l'ensemble d'individus. Après cette sélection, on affecte chaque individu au centre le plus proche en créant k classes, les centres des classes seront remplacés par les centres de gravité et de nouvelles classes seront créés par

le même principe.

Généralement la partition obtenue est localement optimale car elle dépend du choix initial des centres. Pour cela les résultats entre deux exécutions de l'algorithme sont significativement.

c) Méthode de nuées dynamiques

Dans ce cas, le problème posé est la recherche d'une partition en **k** (k fixé) classes d'un ensemble de n individus. C'est un algorithme itératif.

Soit **I** une population d'individus, cette population est représentable sur **R** et forme un nuage de **n** points.

On cherche à constituer une partition en **k** classes sur **i**. chaque classe est représentée par son centre, également appelé noyau, constitué du petit sous-ensemble de la classe qui minimise le critère de dissemblance. [33]

La **figure 2.13** suivante donne un aperçu de l'application des différentes étapes d'algorithme des **k-means** pour le partitionnement (clustering) en **2** classes (***k*** = **2**) :

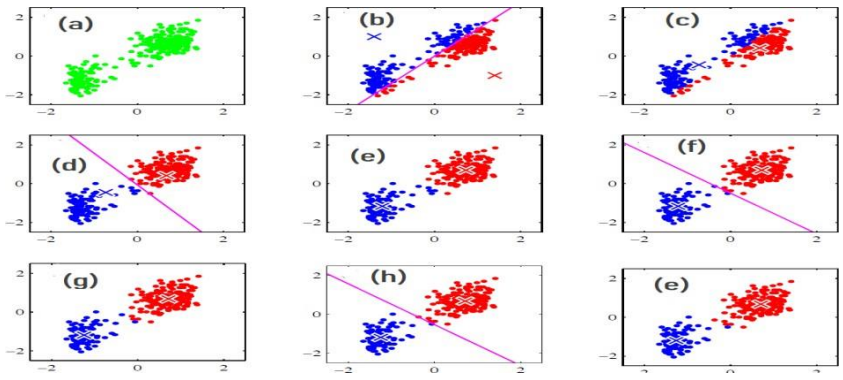


Figure 2.13: Aperçu de l'algorithme K-means

d) La classification hiérarchique ascendante

La CAH offre la possibilité de créer une hiérarchie complète des objets sous forme d'un « arbre » dans un ordre croissant. On débute en envisageant chaque individu comme une classe et on essaye de fusionner deux ou plusieurs classes appropriées (selon une similarité) pour former une nouvelle classe. Le processus est étiré jusqu'à ce que tous les individus se trouvent dans une même classe. Cette classification génère un arbre que l'on

peut couper à différents niveaux pour obtenir un nombre de classes plus ou moins grand. Différentes mesures de la distance inter classes peuvent être utilisées : la distance euclidienne, la distance inférieure (qui favorise la création de classes de faible inertie) ou la distance supérieure (qui favorise la création de classes d'inertie plus importante) etc. Dans le cas de la classification ascendante hiérarchique, à partir des éléments, on forme des petites classes ne comprenant que des individus très semblables, puis à partir de celle-ci, on construit des classes de moins en moins homogènes, jusqu'à obtenir la classe tout entière. [34]

Un exemple classification hiérarchique ascendante est représenté ci-dessous :

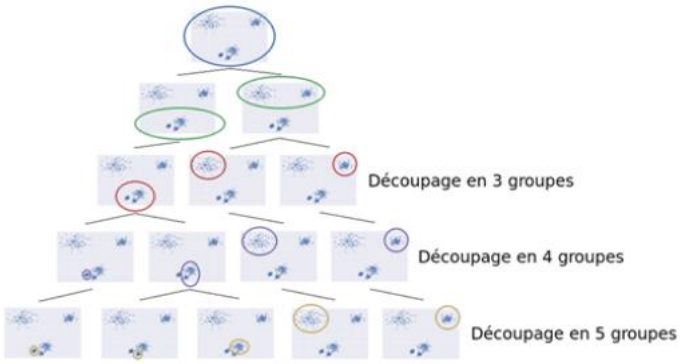


Figure 2.14: Classification hiérarchique ascendante

e) Algorithme FCM (Fuzzy C-Means)

FCM (Fuzzy C-Means) est un algorithme de classification non supervisée floue. Issu de l'algorithme des C-moyennes (C-means), il introduit la notion d'ensemble flou dans la définition des classes : Chaque point dans l'ensemble de données est associé à chaque regroupement avec un certain niveau de correspondance, et tous les regroupements sont définis par leur centre de masse. À l'instar des autres algorithmes de classification non supervisée, il se base sur un critère de minimisation. On travaille sur la minimisation des distances intra-classe et la maximisation des distances inter-cluster, tout en attribuant un certain niveau d'appartenance à chaque classe pour chaque pixel. Cet algorithme requiert une connaissance préalable du nombre de regroupements et crée les classes grâce à un processus itératif visant à minimiser une fonction objective. Ainsi, il permet d'obtenir une partition floue de l'image en donnant à chaque pixel un degré d'appartenance

(compris entre 0 et 1) à une classe donnée. Le cluster auquel est associé un pixel est celui dont le degré d'appartenance sera le plus élevé. [35]

FCM est basée sur la minimisation de la fonction objective définie par:

$$J(U, c_1, c_2, c_3) = \sum_{i=1}^{c} \sum_{j}^{n} \cup_{ij}^{m} \mathbf{d_{ij}^2} \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \textbf{(2.7)}$$

Et m étant un paramètre constant supérieur à 1 (généralement pris égal à 2), $\boldsymbol{U_{ij}}$est la valeur d'appartenance du nœud j au cluster i, c est le nombre de clusters, n est le nombre de nœuds et dij est la distance euclidienne entre le nœud j et le centre du cluster [36]

2.12 Algorithme de rétropropagation du gradient de l'erreur

L'algorithme de rétropropagation du gradient d'erreur est un processus itératif dont le but est de déterminer les poids des connexions afin de réduire l'erreur quadratique moyenne générée par le réseau sur l'ensemble d'apprentissage. L'application d'une méthode de gradient pour minimiser conduit à l'algorithme d'apprentissage par rétropropagation. Cet algorithme, qui a le mérite d'exister, demeure contestable étant donné que sa convergence n'est pas démontrée. Son emploi pourrait entraîner des blocages dans un minimum local de la surface d'erreur. En effet, son efficacité est conditionnée par un vaste ensemble de paramètres que l'utilisateur doit déterminer : le taux d'apprentissage, les réglages des fonctions d'activation, la structure du réseau, le nombre de couches, le nombre de neurones par couche, l'initialisation des poids, etc.

2.12.1 Etapes de l'algorithme de rétropropagation

$\boldsymbol{W_{ik}}$: La connexion entre le neurone ***i*** et le neurone ***k***.

$\mathbf{W_{kh}}$: La connexion entre le neurone ***k*** et le neurone ***h***

$\mathbf{net_i}$: L'entrée totale du neurone ***i*** de la couche (3).

$\mathbf{net_k}$: L'entrée totale du neurone ***k*** de la couche (2).

$\boldsymbol{\delta_i}$: L'erreur quadratique

$\boldsymbol{\alpha}$: Pas d'apprentissage

Y : La sortie

$\boldsymbol{\theta}$: Le seuil (le biais)

Dans ce qui suit nous allons résumer les différentes étapes de la règle de la rétropropagation :

1) Appliquer un vecteur d'entrée, aux unités d'entrée :

$$\mathbf{X} = (\mathbf{X_1}, \mathbf{X_2}, \mathbf{X_3}, \ldots, \mathbf{X_N}) \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \textbf{(2.8)}$$

2) Calculer les entrées totales des neurones cachés :

$$\mathbf{net_k^{(2)} = \sum_{j=1}^{p} W_{kh}\, X_j + \theta_k^{(2)}} \quad \textbf{(2.9)}$$

3) Calculer les sorties des neurones cachés :

$$\mathbf{Y_k^{(2)} = F\left(net_k^{(2)}\right)} \quad \textbf{(2.10)}$$

4) Calculer les entrées totales des neurones de la couche de sortie :

$$\mathbf{net_i^{(2)} = \sum_{k=1}^{n} W_{ik}\, Y_k + \theta_i^{(2)}} \quad \textbf{(2.11)}$$

5) Calculer les sorties réelles des unités de la 3ème couche (couche de sortie) :

$$\mathbf{Y_i^{(3)} = F\left(net_i^{(3)}\right)} \quad \textbf{(2.12)}$$

6) Calculer les termes d'erreur sur chaque unité de sortie

$$\mathbf{\delta_i^{(3)} = Y_i^{(3)} - Yd_i^{(3)}} \quad (\mathbf{2.13})$$

7) Calculer les termes d'erreur sur chaque unité cachée :

$$\mathbf{\delta_k^{(3)} = F\left(net_k^{(2)}\right) \sum_{i=1}^{n} \delta_i^{(3)} W_{kh}^{(3)}} \quad \textbf{(2.14)}$$

Notons que les termes d'erreur pour les unités de la couche cachée sont calculés avant que les connexions des poids aux unités de la couche cachée n'aient été mises à jour. Ainsi, l'erreur est rétropropagée de la sortie vers l'entrée, d'où le nom rétropropagation.

8) Mettre à jour les poids de la couche de sortie :

$$\mathbf{W_{ik}^{(3)}(t+1) = W_{ik}^{(3)}(t) - \alpha \delta_i^{(3)} Y_k^{(2)}} \quad \textbf{(2.15)}$$

9) Mettre à jour les poids de la couche cachée :

$$\mathbf{W_{kh}^{(2)}(t+1) = W_{kh}^{(2)}(t) - \alpha \delta_k^{(2)} Y_h^{(1)}} \quad \textbf{(2.16)}$$

Répéter ce processus jusqu'à ce que l'erreur soit atteinte pour tous les exemples, c'est a dire lorsqu'elle devienne suffisamment petite pour chaque paire d'apprentissage

2.13 Métriques de mesure de performance d'un classifieur

2.13.1 Précision des model d'apprentissage

La précision est une mesure couramment utilisée pour évaluer les performances des modèles de classification en apprentissage automatique. Elle offre une indication de la capacité du modèle à classer correctement les instances positives et négatives.

La simplicité de la précision en fait une métrique largement comprise par les utilisateurs finaux et les scientifiques des données.

Cependant, il est important de reconnaître que la précision présente certaines limites. En se concentrant uniquement sur cette métrique, on peut négliger les nuances des erreurs commises par les modèles d'apprentissage automatique.

La précision ne tient compte que du nombre de prédictions correctes par rapport au nombre total de prédictions, sans considérer les faux positifs et les faux négatifs. Par conséquent, un modèle avec une précision élevée peut encore présenter des erreurs significatives, telles que des faux positifs ou des faux négatifs.

Il est donc important de prendre en compte d'autres métriques d'évaluation, telles que le rappel, qui fournit une vision plus complète des performances du modèle. Ces métriques permettent de considérer à la fois les erreurs positives et négatives, offrant une évaluation plus approfondie et nuancée des performances du modèle de classification.

En résumé, bien que la précision soit une métrique simple et largement utilisée, il est nécessaire de l'interpréter avec prudence et de compléter son évaluation par d'autres métriques pour obtenir une vision plus complète des performances du modèle d'apprentissage automatique.

- **La précision**

La précision indique combien du nombre total de prédictions spécifiées comme positives sont correctement affectées. Cette métrique est également connue sous le nom de valeur prédictive positive.

Example:

		Predicted	
		1	0
Actual	1	TP	FN
	0	FP	TN

Dans la matrice de confusion ces métriques sont calculées de la manière suivante :

$$\textbf{Précision} = \frac{\textbf{TP}}{(\textbf{TP+FP})} \quad \textbf{(2.17)}$$

- **Le Rappel :** Le rappel est le nombre total de cas positifs réels qui ont été correctement prédits.

$$\textbf{Rappel} = \frac{\textbf{TP}}{(\textbf{TP+FN})} \text{.....(2.18)}$$

- **La mesure F1 :** La mesure F1 combine précision et rappel. Le résultat est la moyenne harmonique des deux valeurs, il est calculé comme suit :

$$\textbf{F1} = \frac{\textbf{2}\times(\textbf{Précision}\times\textbf{Rappel})}{(\textbf{Précision+Rappel})} \text{ ..... (2.19)}$$

Soit un exemple ci dessous d'un modèle qui prédit sur la base de certains attributs physiques et comportementaux si un animal était un chien ou un chat.

Example

		Predicted: Dog	Predicted: Cat	
Actual	Dog	24	6	30
	Cat	2	18	20

Voici les résultats si "**Chien**" est utilisé comme réponse positive.

- **Précision** = 24 ÷ (24 + 2) = 0,9231
- **Rappel** = 24 ÷ (24 + 6) = 0,8
- **F1** = 2 × (0,9231 × 0,8) ÷ (0,9231 + 0,8) = 0,8572

Comme on peut le voir, la valeur F1 se situe entre les valeurs de précision et de rappel.
Bien que la précision F1 ne soit pas aussi facile à comprendre, elle ajoute de la nuance au nombre de précision de base.

2.13.2 Over all Accuracy (précision global)

La précision globale est la probabilité qu'un individu soit correctement classé par un test, c'est-à-dire la somme des vrais positifs et des vrais négatifs divisés par le nombre total d'individus testés.

$$\textbf{OA} = \frac{\textbf{nombre déchontillons correctement classé}}{\textbf{nombre déchontillons}} \text{.....(2.20)}$$

Example:

		Validation		
		Forest	Non-For.	*Sum*
Class.	Forest	6	0	*6*
	Non-For.	2	1	*3*
	Sum	*8*	*1*	*9*

$$OA = \frac{6+1}{9} = 0.78 = 78\%$$

2.13.3 Average Accuracy (AA)

Elle calcule l'exactitude moyenne par classe en prenant la moyenne de l'exactitude de chaque classe individuelle. Sa formule est:

$$\mathbf{AA} = \frac{1}{N}\sum_{i=1}^{N}\frac{\text{Prédictions correctes pour la classe i}}{\text{Total d'prédictions de la classe i}} \quad \text{.....(2.21)}$$

Où N est le nombre total de classes.

L'Overall Accuracy donne une vue d'ensemble des performances globales du modèle, tandis que l'Average Accuracy peut être utile pour comprendre l'équilibre des prédictions entre les différentes Classes. Dans le cas où certaines classes sont sur-représentées, l'AA peut être une meilleure métrique que l'OA, car elle évite un biais en faveur des classes majoritaires.

Ces métriques sont particulièrement importantes pour évaluer des tâches comme la classification d'images, notamment avec des datasets comme MNIST ou SVHN.

2.13.4 Le coefficient Kappa

Le coefficient Kappa est calculé à partir d'un test statistique utilisé pour évaluer l'exactitude d'une classification. Il mesure essentiellement la performance de la classification par rapport à une attribution aléatoire des valeurs, c'est-à-dire s'il y a une amélioration par rapport au hasard. Le coefficient Kappa peut varier de -1 à 1. Une valeur de 0 indique que la classification n'est pas meilleure qu'une classification aléatoire. Une valeur négative indique que la classification est significativement pire que le hasard. Une valeur proche de 1 indique que la classification est significativement meilleure que le hasard.

Tableau 2.3 : Interprétation de K

K	Interprétation
< 0	Désaccord
0,00 – 0,20	Accord très faible
0,21 – 0,40	Accord faible
0,41 – 0,60	Accord modéré
0,61 – 0,80	Accord fort
0,81 – 1,00	Accord presque parfait

Il fournit une indication de la fiabilité de la classification et permet de determiner si la performance est significativement meilleure ou pire que ce qui serait attendu par hasard. Plus la valeur de Kappa est proche de 1, plus la classification est considérée comme fiable et meilleure que le hasard.

Il convient de noter que le coefficient Kappa doit être utilisé avec prudence et dans le contexte approprié. Il est recommandé de l'interpréter conjointement avec d'autres métriques d'évaluation pour obtenir une évaluation complète des performances du modèle de classification.

$$\mathbf{Kappa} = \frac{\mathrm{N}\sum_{\mathrm{i}=1}^{\mathbf{n}} \mathrm{M}_{\mathrm{i,i}} - \sum_{\mathrm{i}=1}^{\mathbf{n}} [\![(\mathrm{G}]\!]_{\mathrm{i}} \mathrm{C}_{\mathrm{i}})}{\mathbf{N^2} - \sum_{\mathrm{i}=1}^{\mathbf{n}} [\![(\mathrm{G}]\!]_{\mathrm{i}} \mathrm{C}_{\mathrm{i}})} \quad \textbf{(2.21)}$$

Où :

- **i** est le numéro de classe.
- $\boldsymbol{N}$ est le nombre total de valeurs classer.
- $\mathbf{M_{i,i}}$ est le nombre de valeurs appartenant à la classe de vérité $\boldsymbol{i}$qui ont également été classées dans la classe **i** (c'est-à-dire les valeurs trouvées le long de la diagonale de la matrice de confusion).
- $\boldsymbol{C_I}$ est le nombre total de valeurs prédites appartenant à la classe **i**.
- $\boldsymbol{G_i}$ est le nombre total de valeurs de vérité appartenant à la classe **i**.

2.14 Algorithme d'optimisation

2.14.1 Descente du gradient (Gradient Descent)

La descente de gradient est un algorithme d'optimisation utilisé en apprentissage automatique pour minimiser la fonction de coût $\mathbf{J(\theta)}$ d'un modèle en ajustant itérativement ses paramètres $\boldsymbol{\theta}$ dans la direction opposée du gradient.

Le gradient $\boldsymbol{\nabla(\theta)}$ représente la pente de la fonction de coût, et en mettant à jour les paramètres dans la direction du gradient négatif, l'algorithme peut converger vers l'ensemble optimal de paramètres qui correspond le mieux aux données d'apprentissage.

L'équation générale de mise à jour des paramètres pour la descente de gradient est :

$$\boldsymbol{\theta_{t+1} = \theta_t - \alpha \nabla J(\theta_t)} \quad \textbf{(2.22)}$$

$\boldsymbol{\theta_t}$: représente les paramètres du modèle à l'itération t.

$\boldsymbol{\alpha}$: est le taux d'apprentissage (taille du pas) .

$\boldsymbol{\nabla(\theta t)}$: est le gradient de la fonction de coût par rapport aux paramètres.

Il existe différentes variantes de descente de gradient :

2.14.2 Descente de gradient par lots (Batch Gradient Descent)

Dans la descente de gradient par lots, le gradient est calculé à l'aide de l'ensemble des données d'apprentissage. L'équation de mise à jour des paramètres devient :

$$\boldsymbol{\theta_{t+1} = \theta_t - \alpha \frac{1}{N} \sum_{i=1}^{N} \nabla J\left(\theta_t, x^{(i)}, y^{(i)}\right)} \quad \textbf{(2.23)}$$

Où **N** est le nombre total d'exemples d'apprentissage et (($\boldsymbol{i}$), $\boldsymbol{y(i)}$) représente le ***ième*** exemple d'apprentissage.

2.14.3 Descente de gradient stochastique (Stochastic Gradient Descent)

Dans la descente de gradient stochastique, le gradient est calculé en utilisant un seul exemple d'apprentissage à la fois. L'équation de mise à jour des paramètres devient :

$$\boldsymbol{\theta_{t+1} = \theta_t - \alpha \nabla J\left(\theta_t, x^{(i)}, y^{(i)}\right)} \quad \textbf{(2.24)}$$

2.14.4 Descente de gradient en mini-lot (Mini-Batch Gradient Descent))

La descente de gradient en mini-lot est un compromis entre la descente de gradient par lot et stochastique. Il calcule le gradient à l'aide d'un petit lot d'exemples de formation [36]. L'équation de mise à jour des paramètres devient:

$$\theta_{t+1} = \theta_t - \alpha \frac{1}{B} \sum_{i=1}^{B} \nabla J\left(\theta_t, x^{(i)}, y^{(i)}\right) \dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots (2.25)$$

Où **B** est la taille du mini-lot et $\{(x^{(i)}, y^{(i)})\}_{i=1}^{B}$ représente le mini-lot d'exemples d'apprentissage.

2.14.5 Mini Batch - Descente de gradient stochastique

Le MB-SGD représente une avancée de l'algorithme de descente de gradient stochastique (SGD), qui est couramment employé pour minimiser une fonction objectif formulée comme une somme de fonctions différentiables (**voir Figure 2.15**). Cette méthode permet de pallier la lenteur inhérente à l'algorithme SGD en sélectionnant un sous-ensemble de points de l'ensemble de données pour le calcul du gradient. Récemment, les algorithmes d'optimisation adaptative ont gagné en popularité grâce à leur aptitude à converger rapidement, en s'appuyant sur des statistiques issues des itérations antérieures pour optimiser le processus de convergence **[14].**

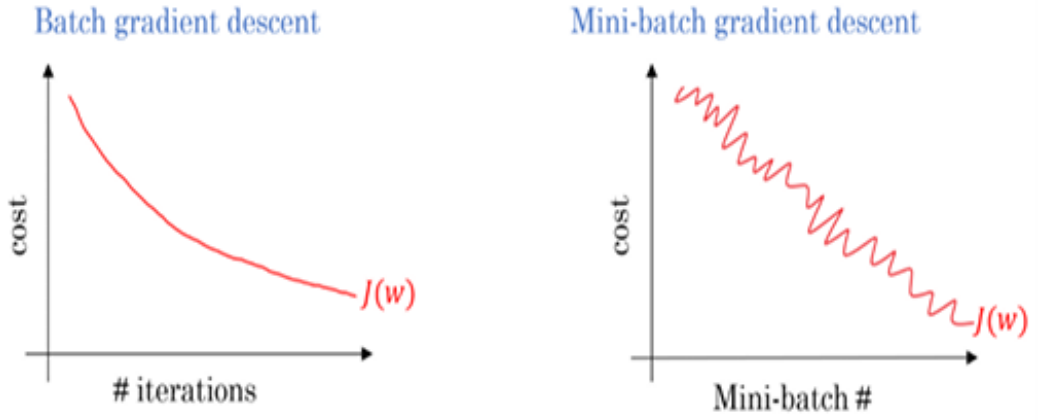


Figure 2.15 : Le gradient descente et le mini batch de gradient descente.

- **Calcul du gradient pour un mini-batch** :

$$g_t = \frac{1}{B} \sum_{i=1}^{B} \nabla J(\theta, x_i, y_i) \dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots (2.26)$$

Où:

- **B** est la taille du mini-batch.
- **J (θ, x_i, y_i)** est la fonction de coût pour l'exemple **i**.
- g_t Est le gradient moyen calculé sur le mini-batch.
- **Mise à jour des paramètres** :

$$\theta_t = \theta_{t-1} - \alpha g_t \quad \text{(2.27)}$$

Où :

- **α** est le taux d'apprentissage.
- **θ_t** représente les paramètres du modèle à l'étape **t**.

2.14.6 Optimiseur basé sur Momentum

Cet algorithme d'optimisation adaptative exploite de manière exponentielle les gradients moyens pondérés des itérations antérieures afin de stabiliser le processus de convergence, ce qui permet d'accélérer l'optimisation. Ce mécanisme est réalisé en intégrant une fraction (gamma) aux valeurs issues de l'itération précédente. Fondamentalement, le terme d'élan s'intensifie lorsque les points de gradient s'alignent dans la même direction et s'atténue lorsque les gradients varient. Par conséquent, la convergence de la valeur de la fonction de perte s'effectue plus rapidement que prévu **[15]**.

- Mise à jour de l'accumulation du gradient :

$$m_t = \beta m_{t-1} + g_t \quad \text{(2.28)}$$

- **Mise à jour du paramètre** :

$$\theta_t = \theta_{t-1} - \alpha m_t \quad \text{(2.29)}$$

Où :

- **m_t** est le **terme de momentum**, qui agit comme une "mémoire" des gradients passés, Vecteur d'accumulation des gradients.
- **β** est le **coefficient de momentum** (généralement proche de 0.9).
- g_t est le **gradient** de la fonction de coût à l'itération t.
- **α** est le **taux d'apprentissage**.

Le principe derrière **Momentum** est d'utiliser une moyenne pondérée des gradients passés pour stabiliser la convergence et accélérer l'optimisation. Il permet de réduire les oscillations et

d'améliorer l'apprentissage, notamment dans des architectures complexes comme les **réseaux de neurones convolutionnels (CNN)**.

2.14.7 RMSProp

Le Root Mean Squared Prop (RMSProp) constitue une méthode alternative d'adaptation du taux d'apprentissage, visant à perfectionner l'algorithme AdaGrad. Contrairement à AdaGrad, qui utilise la somme cumulative des gradients au carré, RMSProp opte pour une moyenne mobile exponentielle. Les étapes initiales de ces deux méthodes sont identiques, mais RMSProp se distingue en divisant le taux d'apprentissage par une moyenne qui décroît de manière exponentielle **[16]**.

- **Calcul de la moyenne mobile des carrés des gradients :**

$$\mathbf{v_t = \beta v_{t-1} + (1-\beta) g_t^{\,2}} \quad \textbf{(2.30)}$$

- **Mise à jour du paramètre :**

$$\mathbf{\theta_t = \theta_{t-1} - \frac{\alpha}{\sqrt{v_t + \varepsilon}} g_t} \quad \textbf{(2.31)}$$

Où:

- $\mathbf{v_t}$ est la moyenne mobile des carrés des gradients.
- $\boldsymbol{\beta}$ est le **facteur de décroissance** (souvent autour de 0.9).
- $\mathbf{g_t}$ représente le gradient de la fonction de coût.
- $\boldsymbol{\alpha}$ est **le taux d'apprentissage.**
- $\boldsymbol{\epsilon}$ est un **petit terme** (souvent 10^{-8}) pour éviter la division par zéro.

Cet algorithme ajuste automatiquement le taux d'apprentissage en fonction des gradients, ce qui est particulièrement efficace dans **les réseaux de neurones convolutionnels (CNN)** où les gradients peuvent fluctuer fortement.

2.14.8 Estimation adaptative du moment (Adam)

Il s'agit d'une méthode qui allie les principes de RMSProp et de Momentum. Cette approche permet de déterminer un taux d'apprentissage adaptatif pour chaque paramètre. En plus de conserver la moyenne décroissante des carrés des gradients précédents, elle intègre également la moyenne des gradients antérieurs, à l'instar de Momentum. Par conséquent, Adam agit comme une masse lourde soumise à un frottement, favorisant ainsi les minimas plats sur la surface de l'erreur.

L'**Estimation Adaptative du Moment (Adam)** est un optimiseur qui combine les avantages de **Momentum** et **RMSProp** pour ajuster dynamiquement le taux d'apprentissage dans les réseaux de neurones convolutionnels (**CNN**). Il utilise deux moyennes exponentielles des gradients pour stabiliser et accélérer la convergence **[17].**

L'équation mathématique de **Adam** est définie comme suit :

- **Calcul des moments exponentiels:**

$$\mathbf{m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t} \quad \textbf{(2.32)}$$

- **Calcul de la moyenne mobile des carrés des gradients** (second moment) :

$$\mathbf{v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2} \quad \textbf{(2.33)}$$

Où :

- $\mathbf{g_t}$ représente le gradient du paramètre $\boldsymbol{\theta_t}$ à l'étape **t**.
- $\mathbf{m_t}$ et $\mathbf{v_t}$ sont les estimations du premier et du second moment des gradients.
- $\boldsymbol{\beta_1}$ et $\boldsymbol{\beta_2}$ sont les coefficients de décroissance (souvent $\beta_1 = 0.9$ et $\beta_2 = 0.999$).

3.4 Conclusion

Dans ce chapitre, les réseaux neuronaux convolutifs sont aujourd'hui une solution essentielle pour la classification d'images et l'extraction de caractéristiques à partir de données visuelles. Leur architecture en couches hiérarchiques leur permet de repérer des motifs de plus en plus complexes à mesure que l'on avance dans le réseau. Ce chapitre a souligné leur performance, leurs différentes variantes, leurs méthodes d'apprentissage, ainsi que les techniques d'optimisation modernes telles qu'Adam ou RMSProp. Enfin, les différentes stratégies de classification et les métriques d'évaluation permettent d'évaluer la performance des modèles dans divers contextes. L'évolution permanente des CNN ouvre la voie à des applications toujours plus avancées dans des secteurs variés comme la santé, la sécurité, la robotique ou encore la traduction automatique.

CHAPITRE 3 : LES DIFFERENTES FONCTIONS D'ACTIVATION DES RESEAUX NEURONES ET LEUR IMPACT SUR LES PERFORMANCES DE DIVERS MODELES CNN POUR LA CLASSIFICATION

CHAPITRE 3 : LES DIFFERENTES FONCTIONS D'ACTIVATION DES RÉSEAUX NEURONES ET LEUR IMPACT SUR LES PERFORMANCES DE DIVERS MODÈLES CNN POUR LA CLASSIFICATION

3.1 Introduction

Les fonctions d'activation sont essentielles pour le fonctionnement des réseaux de neurones artificiels. Elles introduisent la non-linéarité nécessaire pour capturer des relations complexes dans les données. Ce chapitre décrit les principales fonctions d'activation employées dans les architectures modernes de deep learning, notamment dans les réseaux de neurones convolutifs (CNN). Il examine leurs caractéristiques, leurs avantages, leurs inconvénients, ainsi que leur impact sur la convergence du modèle. À travers des comparaisons détaillées, ce chapitre met en évidence combien le choix de la fonction d'activation influence la performance et la stabilité des réseaux neuronaux.

3.2 Definition des fonctions d'activations

Une fonction d'activation est une fonction mathématique utilisé sur un signal. Elle va reproduire le potentiel d'activation que l'on retrouve dans le domaine de la biologie du cerveau humain. Elle va permettre le passage d'information ou non de l'information si le seuil de stimulation est atteint. Concrètement, elle va avoir pour rôle de décider si on active ou non une réponse du neurone. Un neurone (**voir Figure 3.1**) ne va faire qu'appliquer la fonction suivante :

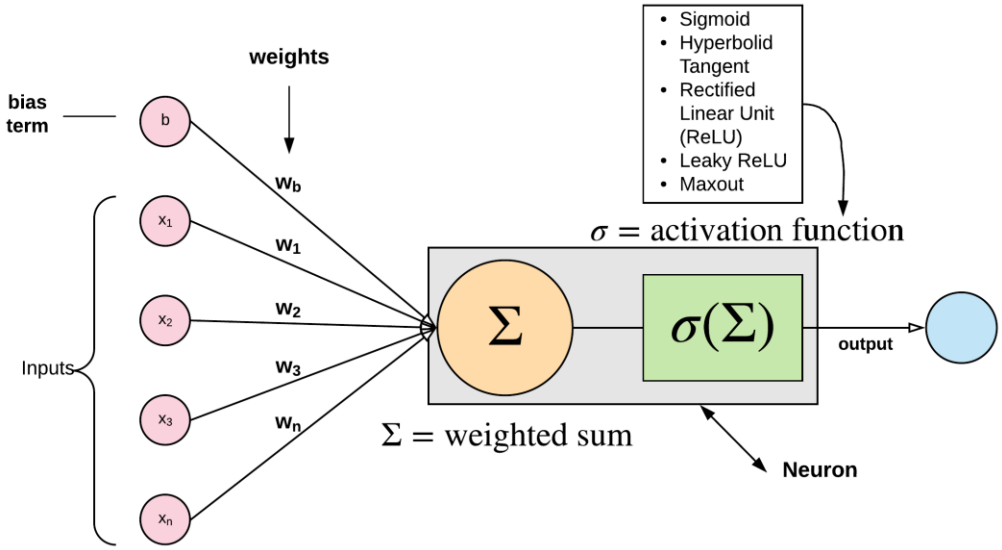


Figure 3.1: Un neuron

CHAPITRE 3 : LES DIFFERENTES FONCTIONS D'ACTIVATION DES RÉSEAUX NEURONES ET LEUR IMPACT SUR LES PERFORMANCES DE DIVERS MODÈLES CNN POUR LA CLASSIFICATION

$$\textbf{sortie} = \sigma(\sum(\textbf{entrée} * \textbf{poids}) + \textbf{biais}) \quad \textbf{(3.1)}$$

3.3 Pourquoi utiliser des fonctions d'activation dans un réseau neuronal

En référence à l'exemple ci-dessus **(voir Figure 3.2)**, la relation entre les poids et les entrées neuronales est toujours linéaire si nous n'utilisons pas de fonction d'activation. Ainsi, la sortie est une simple transformation linéaire de ces derniers. Lorsque seule la relation linéaire entre les variables est abordée, le modèle prévu ne convient pas pour des problèmes complexes tels que la vision par ordinateur et le traitement du langage naturel.

Ici, l'utilisation d'une fonction d'activation qui fournit la non-linéarité devient un must. Cela nous permet de construire des modèles plus complexes qui correspondent aux problèmes du monde réel **[37].**

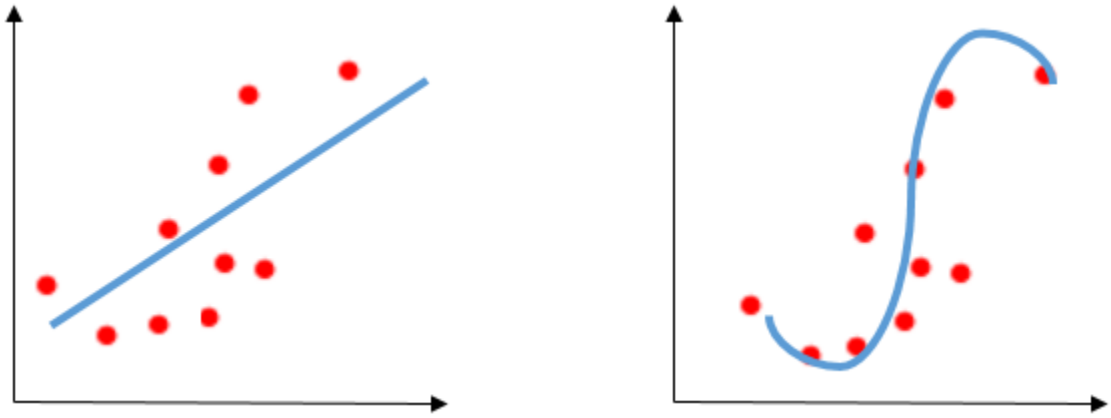


Figure 3.2 : La relation entre les poids et les entrées neuronales.

3.4 Comparaison de différentes fonctions d'activation de réseaux de neurones convolutifs et de procédés de construction d'ensembles

Voici les principales fonctions d'activations que l'on peut trouver dans le domaine des réseaux de neurones :

3.4.1 SOFTPLUS

La fonction d'activation Softplus est une version lisse de la fonction ReLU, définie par :

$$\mathbf{f(x) = \ln(1 + e^{x})} \text{.....(3.2)}$$

Elle est toujours positive et différentiable, ce qui la rend utile pour stabiliser l'entraînement des réseaux de neurones convolutifs (CNN) (**voir Figure 3.3)[37]**.

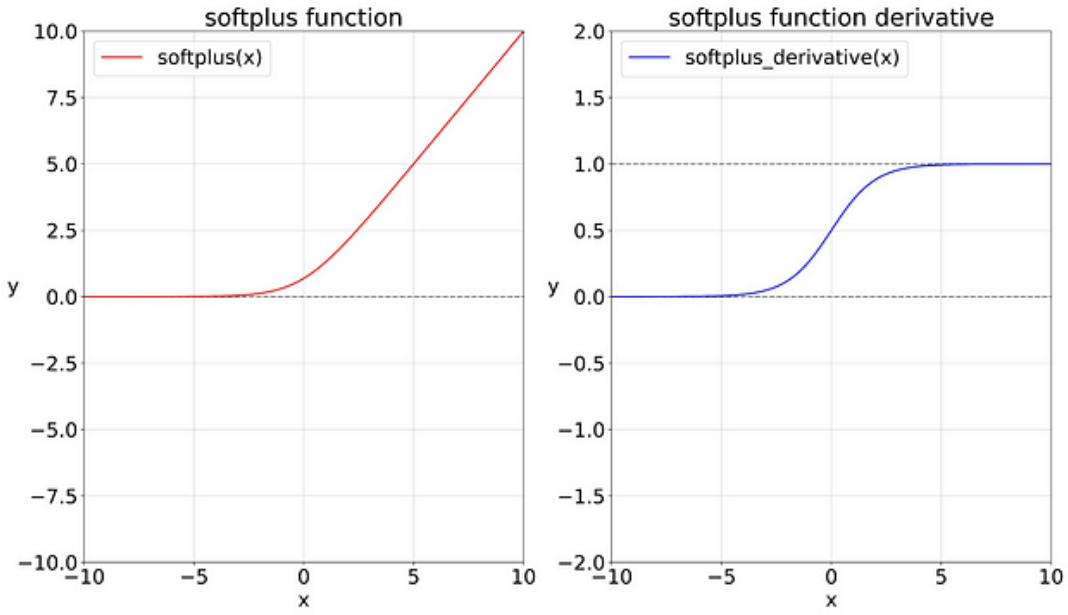


Figure 3.3 : La fonction d'activation Softplus et la derivé.

- **Avantages**

- **Différentiabilité complète** : Contrairement à ReLU, Softplus est entièrement différentiable, ce qui évite les problèmes de gradient nul.
- **Lissage du paysage de perte** : Elle réduit les discontinuités et améliore la convergence du modèle.
- **Prévention des valeurs nulles** : Softplus ne produit jamais de sortie strictement nulle, ce qui peut être bénéfique pour certaines architectures **[37]**.

- **Inconvénients**

- **Coût computationnel plus élevé** : Son calcul est plus complexe que ReLU, ce qui peut ralentir l'entraînement.

- **Moins efficace pour les valeurs négatives** : Elle ne conserve pas les valeurs négatives comme Mish, ce qui peut limiter l'expressivité du réseau.
- **Moins populaire** : ReLU reste la norme dans la plupart des architectures CNN **[37]**.

3.4.2 SoftSign

La fonction d'activation **Softsign** est une alternative aux fonctions classiques comme ReLU et Sigmoid, utilisée dans les réseaux de neurones convolutifs **(CNN).** Elle est définie par :

$$\mathbf{f(x) = \frac{x}{1+|x|}} \text{.....(3.3)}$$

Cette fonction est **bornée** entre -1 et 1, ce qui la rend utile pour stabiliser l'entraînement des réseaux neuronaux **(voir Figure 3.4) [39]**.

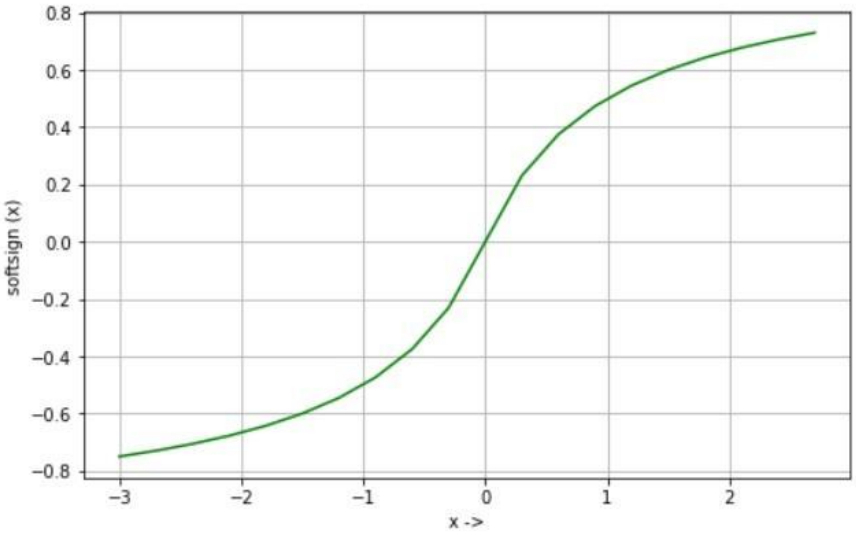


Figure 3.4 : Fonction d'activation SoftSign.

- **Avantages**

- **Lissage du gradient** : Contrairement à ReLU, Softsign offre une transition plus douce, réduisant les oscillations du gradient.
- **Différentiabilité complète** : Elle est entièrement différentiable, ce qui facilite l'optimisation du modèle.
- **Prévention des valeurs extrêmes :** Softsign limite les sorties entre -1 et 1, évitant les activations excessives **[39]**.

- **Inconvénients**

- **Diminution rapide des valeurs élevées :** Pour des entrées très grandes, Softsign tend vers 1, ce qui peut ralentir l'apprentissage.
- **Moins efficace pour les valeurs négatives :** Elle ne conserve pas les valeurs négatives comme certaines autres fonctions d'activation.
- **Moins populaire :** ReLU reste la norme dans la plupart des architectures **CNN [39]**.

3.4.3 GELU

La fonction d'activation GELU (Gaussian Error Linear Unit) est une alternative à ReLU et Softplus, utilisée notamment dans les réseaux de neurones convolutifs (CNN). Elle est définie par :

$$\mathbf{f(x) = x \cdot \Phi(x)} \text{ ……..(3.4)}$$

Où $\Phi(\boldsymbol{x})$ est la fonction de distribution cumulative d'une loi normale standard $\mathbf{\Phi(x) = P(X \leq x)}$ tell que **(X~*N*(0,1)).** Contrairement à ReLU, GELU applique une activation probabiliste qui prend en compte la distribution des valeurs d'entrée **[54]**.

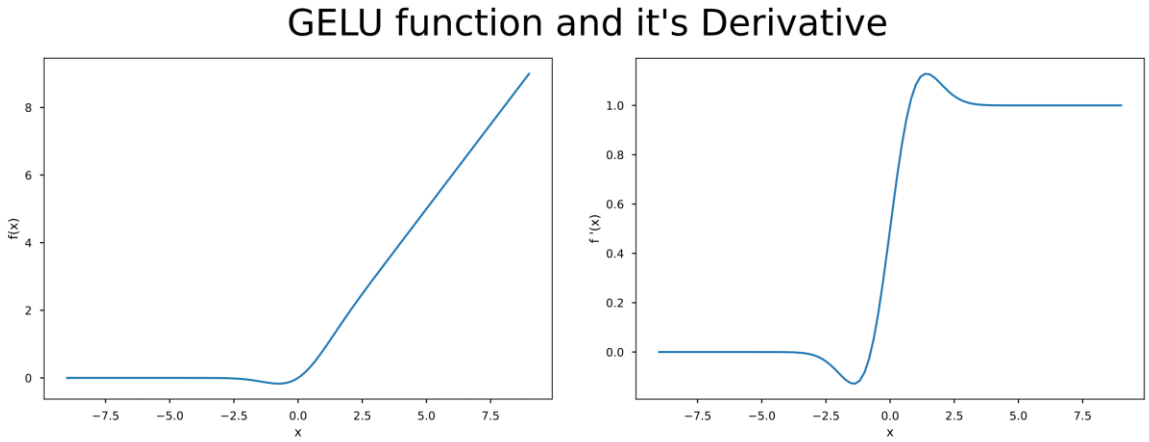


Figure 3.5 : Fonction d'activation GELU et la derivé.

- **Avantages**

- **Meilleure expressivité :** GELU conserve une partie des valeurs négatives, contrairement à ReLU qui les annule complètement.
- **Lissage du paysage de perte :** Son activation douce améliore la convergence et réduit les oscillations du gradient.

- **Performances optimisées** : GELU est utilisé dans des architectures avancées comme Transformers, où il améliore la généralisation du modèle.
- **Différentiabilité complète** : Contrairement à ReLU, GELU est entièrement différentiable, ce qui facilite l'optimisation **[54]**.

- **Inconvénients**

- **Coût computationnel plus élevé :** Son calcul implique une fonction gaussienne, ce qui peut ralentir l'entraînement par rapport à ReLU.
- **Moins répandue :** Bien que populaire dans certains modèles, GELU n'est pas aussi largement adopté que ReLU.
- **Sensibilité aux hyperparamètres :** Son efficacité dépend du réglage du taux d'apprentissage et d'autres paramètres du modèle **[54]**.

3.4.4 Sigmoid (logistic)

Fonction la plus populaire depuis des décennies. Mais aujourd'hui, elle devient beaucoup moins efficace par rapport à d'autre pour une utilisation pour les couches cachées. Elle perd de l'information due à une saturation que cela soit pour la phase de feed forward ou de backpropagation, en donnant des effets non linéaires au réseau due à un paramètre unique. Elle a aussi des soucis de gradient 0 avec des entrées étant très large, même si le souci est minimalisé avec les systèmes utilisant des batch par lots (mini batch). Utilisé en couche de sortie pour de la classification binaire **[38][39]**. Intervalle de sortie $0{,}1$,(**voir Figure 3.6**)

La fonction sigmoïde est donnée par la function :

$$\mathbf{f(x) = sigmoid(x) = \frac{1}{1+e^{-x}}} \quad \textbf{..... (3.5)}$$

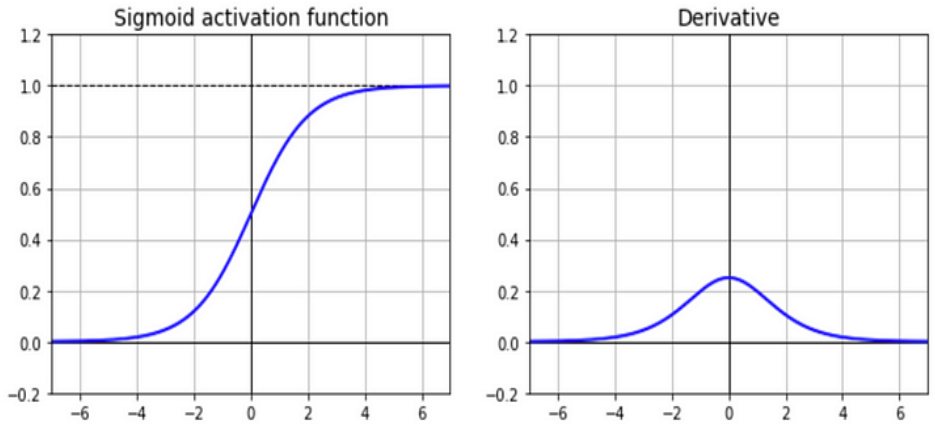


Figure 3.6 : Fonction d'activation sigmoïde et la derivé.

- **Les caractéristique [40]**

- Sortie comprimée entre 0 et 1.
- Historiquement utilisée dans les premiers réseaux de neurones**.**
- Il normalise la sortie du neurone dans une plage comprise entre 0 et 1, donnant la probabilité de la valeur d'entrée. Cela rend le sigmoïde utile pour les neurones de sortie des réseaux de neurones de classification.
- Il est très gourmand en calcul car il nécessite le calcul d'un exposant, ce qui ralentit la convergence du réseau.
- Il souffre d'un problème de saturation. Un neurone est considéré comme saturé s'il atteint sa valeur maximale ou minimale (Ex. Sigmoïde : $f(x) = (\ 0\ ou\ 1\ )$, de sorte que sa dérivée (Ex. Sigmoïde : $f(x)\big(1 - f(x)\big) = 0$) est égale à 0. Dans ce cas, il n'y a pas de mise à jour des poids. Le gradient de la fonction de perte par rapport aux poids disparaît par conséquent jusqu'à ce qu'il descende à 0. Ce phénomène est connu sous le nom de gradient de fuite qui entraîne un mauvais apprentissage des réseaux profonds.
- Ce n'est pas une fonction centrée sur zéro. Ainsi, le gradient de tous les poids connectés au même neurone est soit positif soit négatif. Pendant le processus de mise à jour, ces poids ne peuvent se déplacer que dans une seule direction, c'est-à-dire positive ou négative, à la fois. Cela rend l'optimisation de la fonction de perte plus difficile.

- **Avantage**

- Utile pour les problèmes de classification binaire où une probabilité est souhaitée.

- **Inconvénient:**

- Souffre du problème du **vanishing gradient** lorsque les valeurs sont très grandes ou très petites.
- Moins performante que ReLU dans les architectures profondes.

- **Problèmes avec la fonction d'activation sigmoïde**

Les principaux problèmes liés à la fonction sigmoïde sont **[41]** :

- Dégradé de fuite : En regardant le graphe de la fonction sigmoïde, nous pouvons voir que lorsque les entrées deviennent petites ou grandes, la fonction sature à 0 ou 1, avec une dérivée extrêmement proche de 0. Ainsi, elle n'a presque pas de gradient à se propager à travers le réseau, donc il ne reste presque plus rien pour les couches inférieures.
- Calcul coûteux : la fonction a une opération exponentielle.
- La sortie n'est pas centrée sur zéro**.**

3.4.5 Fonction Switch

Le Swish AF est l'une des premières fonctions d'activation (AF) composées proposées (**voir Figure 3.7**), combinant la fonction sigmoïde et la fonction d'entrée pour obtenir une fonction hybride. L'activation Swish a été proposée par Ramachandran et al., en 2017, et utilise une technique de recherche automatique basée sur l'apprentissage par renforcement pour calculer la fonction. Les propriétés de la fonction Swish incluent la douceur, le caractère non monotone, la limitation inférieure et l'illimitation supérieure. La douceur permet à la fonction Swish de produire de meilleurs résultats d'optimisation et de généralisation lorsqu'elle est utilisée pour entraîner des architectures d'apprentissage profond**.**

La fonction Swish est définie par :

$$\boldsymbol{f(x) = x * sigmoid(x) = \frac{x}{1+e^{-x}}} \text{..... (3.6)}$$

Les auteurs ont souligné que les principaux avantages de la fonction Swish sont sa simplicité et sa précision améliorée. En effet, la fonction Swish ne souffre pas des problèmes de disparition de

gradient, mais assure une bonne propagation de l'information pendant l'entraînement. Ils ont également rapporté que la fonction d'activation Swish surpassait la fonction d'activation ReLU dans les tâches de classification en apprentissage profond **[42].**

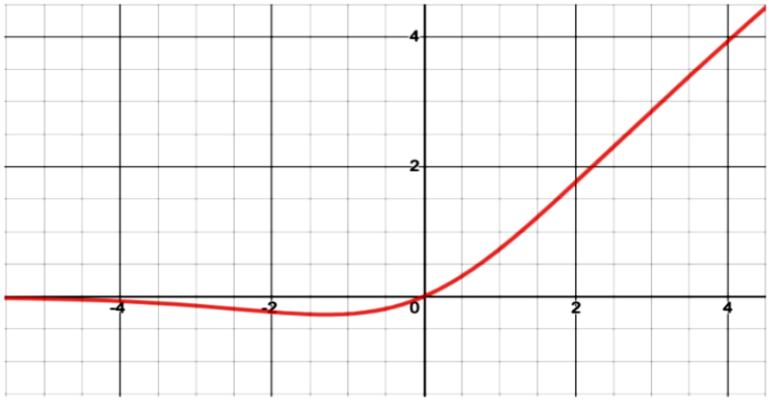


Figure 3.7 : Fonctions d'activation Switch (SiLU).

- **Caractéristiques**

- Introduite par Google en 2017 comme alternative à ReLU.
- Comporte une non-linéarité douce et autorise des gradients non nuls pour toutes les valeurs.

- **Avantages**

- Performances similaires ou meilleures que ReLU dans certains cas.

- **Inconvénients**

- Plus coûteuse en termes de calculs que ReLU.

3.4.6 MISH

Mish est une fonction d'activation neuronale lisse et non monotone, ce qui lui permet de conserver des valeurs négatives et d'améliorer la propagation du gradient qui peut être définie comme suit **[54]**:

$$\mathbf{f(x) = x \cdot \tanh\left(\varsigma(x)\right)} \quad \textbf{.....(3.7)}$$

Où, $\varsigma(x)=\ln(1+e^{x})$ est la fonction d'activation **softplus**. Le graphique de Mish est montré dans la figure (**voir Figure 3.8**).

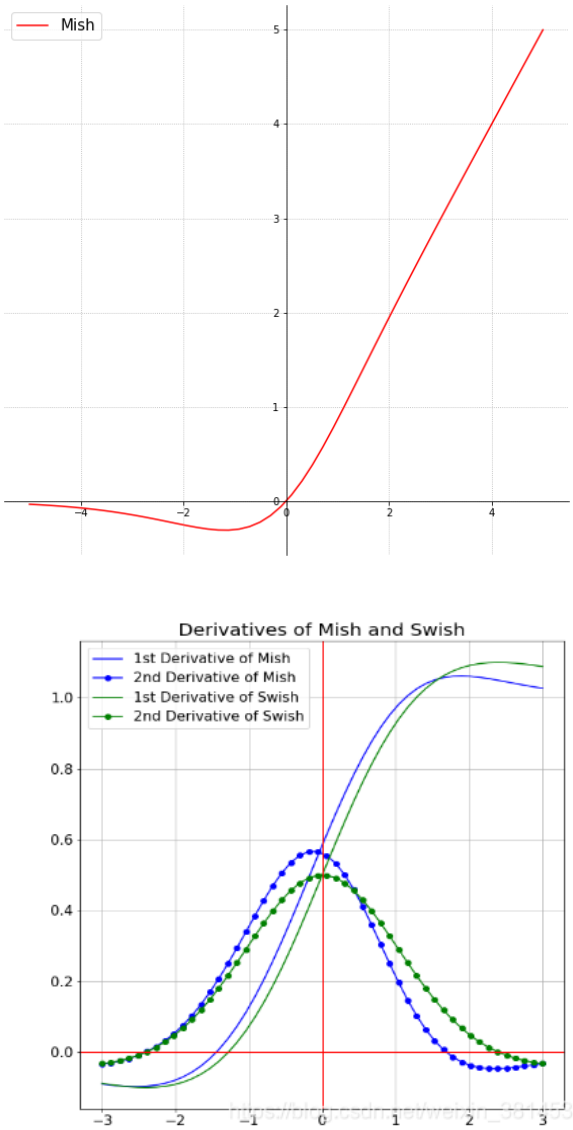


Figure 3.8 : Fonctions d'activation MISH et la derivé.

- **Propriétés de Mish**

Bien qu'il soit difficile d'expliquer la raison pour laquelle une fonction d'activation fonctionne mieux qu'une autre en raison de nombreux autres facteurs d'entraînement, les propriétés de Mish,

telles qu'être non bornée en haut, bornée en bas, lisse et non monotone, jouent toutes un rôle significatif dans l'amélioration des résultats. **La figure (3.9)** montre les différentes fonctions d'activation couramment utilisées ainsi que le graphique de l'activation de Mish pour comparaison **[54]**.

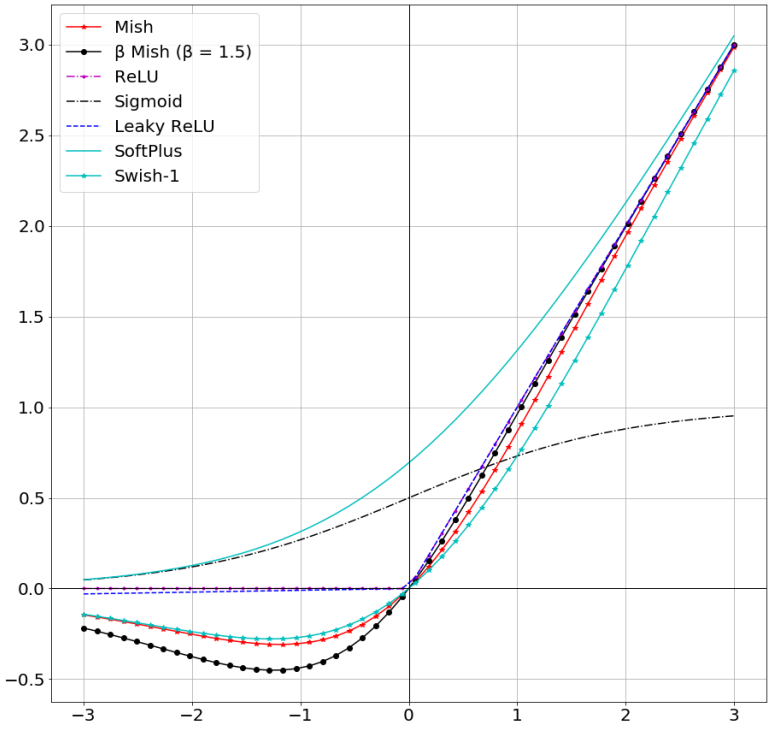


Figure 3.9 : Les différentes fonctions d'activation couramment utilisées

- **Avantages**

- Amélioration de la précision : Mish a montré des performances supérieures à ReLU et Swish sur plusieurs benchmarks, notamment sur des tâches de classification d'images.
- Non-monotonicité : Contrairement à ReLU, Mish conserve les petites valeurs négatives, ce qui améliore l'expressivité du réseau et la propagation du gradient.
- Lissage du paysage de perte : Son caractère continu et différentiable aide à stabiliser l'optimisation et à éviter les problèmes liés aux gradients abrupts.

- Auto-régularisation : Elle offre un effet de régularisation naturel, ce qui peut améliorer la généralisation du modèle. **[54]**

- **Inconvénients**

- Coût computationnel plus élevé : Mish est plus complexe à calculer que ReLU, ce qui peut ralentir l'entraînement, surtout sur des architectures profondes.
- Sensibilité aux hyperparamètres : Elle peut nécessiter un ajustement plus fin du taux d'apprentissage pour éviter des instabilités.
- Moins répandue : ReLU reste la norme dans de nombreux modèles, et Mish n'est pas encore aussi largement adoptée **[54]**.

3.4.7 TanH

Utilisé pour des LSTM (**Long Short-Term Memory**) pour des données en continue.

Intervalle de sortie : $(-\mathbf{1}, \mathbf{1})$

La fonction tangente hyperbolique est un autre type de fonction d'activation utilisée dans l'apprentissage profond (Deep Learning) (**voir Figure 3.10**), et elle possède certaines variantes utilisées dans les applications de **DL**. La fonction tangente hyperbolique, connue sous le nom de fonction **tanh**, est une fonction plus lisse et centrée sur zéro, dont la plage de valeurs se situe entre -1 et 1.

Ainsi, le résultat de la fonction tanh est donné par **[43]** :

$$\mathbf{f(x) = \tan h(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}} \quad \textbf{(3.8)}$$

La fonction tanh est devenue la fonction préférée par rapport à la fonction sigmoïde, car elle offre de meilleures performances d'entraînement pour les réseaux neuronaux à plusieurs couches. Cependant, la fonction tanh n'a pas non plus permis de résoudre le problème de disparition de gradient rencontré par les fonctions sigmoïdes. Son principal avantage réside dans le fait qu'elle génère une sortie **centrée sur zéro**, facilitant ainsi le processus de **rétropropagation**.

Une propriété de la fonction **tanh** est qu'elle ne peut atteindre un gradient de 1 que lorsque la valeur de l'entrée est 0, autrement dit lorsque **x** est **nul**. Cela conduit la fonction tanh à générer des neurones **"morts"** au cours des calculs. Un neurone mort est une condition où le poids d'activation est rarement utilisé en raison d'un gradient nul. Cette limitation de la fonction tanh a encouragé

des recherches supplémentaires sur les fonctions d'activation pour résoudre ce problème, ce qui a donné naissance à la fonction d'activation ReLU (Rectified Linear Unit). Les fonctions tanh ont été principalement utilisées dans les réseaux neuronaux récurrents (RNN) pour le traitement du langage naturel et les tâches de reconnaissance vocale **[43][44]**.

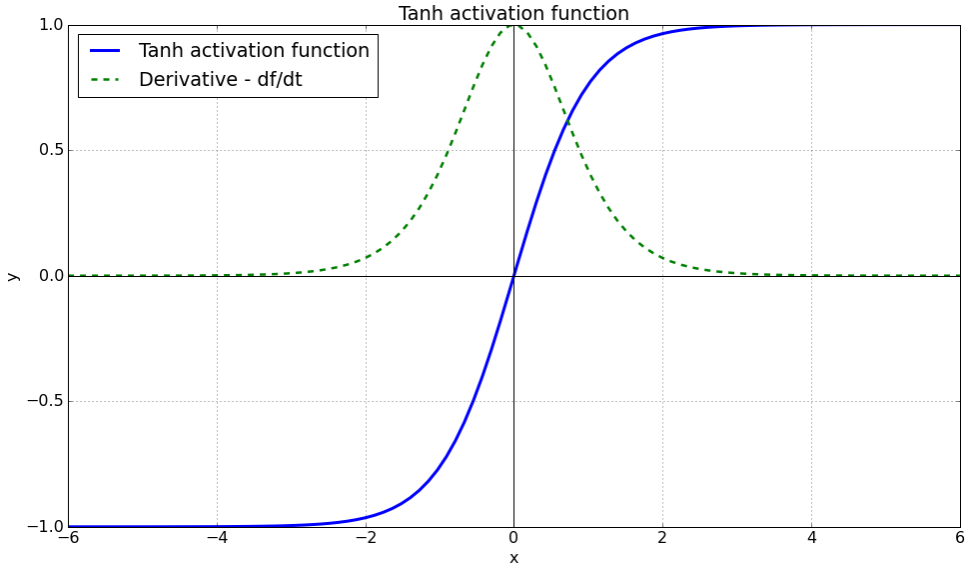


Figure 3.10 : Fonction d'activation Tanh et la derivé.

- **Les caractéristiques [45]**

- Il normalise la sortie du neurone dans une plage comprise entre -1 et 1.
- Sortie comprimée entre -1 et 1.
- Centrée autour de 0, ce qui peut faciliter l'apprentissage
- Contrairement à Sigmoid, il s'agit d'une fonction centrée sur zéro, ce qui facilite l'optimisation de la fonction de perte.
- Quant à Sigmoid, Tanh est très gourmand en calcul et souffre d'un problème de saturation et donc d'un gradient de fuite. En fait, lorsque le neurone atteint la valeur minimale ou maximale de sa plage, qui correspondent respectivement à -1 et 1, sa dérivée est égale à 0.

- **Avantages**

- Meilleure que la fonction sigmoïde pour certaines tâches car elle est centrée autour de 0.

- **Inconvénients**

- Souffre également du problème du (vanishing gradient).

- **Problèmes avec la fonction d'activation de Tanh**

Puisque Tanh a des caractéristiques similaires à Sigmoid, il est également confronté aux deux problèmes suivants **[45]** :

- Dégradé de fuite : en regardant le tracé de la fonction, vous pouvez voir que lorsque les entrées deviennent petites ou grandes, la fonction sature à -1 ou 1, avec une dérivée extrêmement proche de 0. Ainsi, elle n'a presque pas de gradient à se propager à travers le réseau, il ne reste donc presque plus rien pour les couches inférieures.
- Calcul coûteux : la fonction a une opération exponentielle.

3.4.8 Softmax

Utilisé pour de la multi classification en couche de sortie.

Intervalle de sortie$(-\infty, +\infty)$.

La fonction Softmax est un autre type de fonction d'activation utilisée en calcul neuronal. Elle est utilisée pour calculer une distribution de probabilité à partir d'un vecteur de nombres réels. La fonction Softmax produit une sortie qui est une plage de valeurs comprises entre 0 et 1, avec la somme des probabilités égale à 1. La fonction Softmax est calculée à l'aide de la relation.

$$\mathbf{f(x) = \frac{e^{x_i}}{\sum_j e^{x_j}}} \quad \textbf{(3.9)}$$

La principale différence entre les fonctions d'activation Sigmoid et Softmax est que la fonction Sigmoid est utilisée dans les classifications binaires, tandis que la fonction Softmax est utilisée pour les tâches de classification multivariée.

En conclusion, Softmax est utilisé pour la **multi-classification dans le modèle de régression logistique** tandis que Sigmoïde est utilisé pour **la classification binaire dans le modèle de régression logistique**, la somme des probabilités est d'Un 1 pour Softmax **[46][47].**

- **Caractéristiques**

- Généralement utilisée dans les couches de sortie pour les problèmes de classification multi-classe.

- Produit une distribution de probabilités sur plusieurs classes.

- **Avantages**

- Idéale pour les tâches de classification multi-classe.

- **Inconvénients**

- Pas utilisée dans les couches cachées.

3.4.9 ReLU (Rectified Linear Unit)

Ce sont les fonctions les plus populaires de nos jours. Elles permettent un entrainement plus rapide comparé aux fonctions sigmoid et tanh, étant plus légères. Attention au phénomène de 'Dying ReLU', auquel on préférera les variations de ReLU. Plus d'informations en fin d'article. Très utilisé pour les CNN, RBM, et les réseaux de multi perceptron. Intervalle de sortie $(0, +\infty)$ **[48].**
La fonction d'activation unité linéaire rectifiée (ReLU) (**voir Figure 3.11**), a été proposée par Nair et Hinton en 2010 et, depuis, est devenue la fonction d'activation la plus utilisée pour les applications d'apprentissage profond, avec des résultats à la pointe de la technologie à ce jour. La ReLU est une fonction d'activation à apprentissage rapide, qui s'est avérée être la plus réussie et la plus largement adoptée. Elle offre de meilleures performances et une meilleure généralisation en apprentissage profond, comparée aux fonctions d'activation Sigmoid et tanh **[49]**.
La ReLU représente une fonction quasi linéaire et, par conséquent, conserve les propriétés des modèles linéaires qui les rendent faciles à optimiser grâce aux méthodes de descente de gradient. La fonction d'activation ReLU effectue une opération de seuil sur chaque élément d'entrée, où les valeurs inférieures à zéro sont ramenées à zéro **[50]**, ainsi la ReLU est définie par :

$$\mathbf{f(x) = ma\,x(0, x) = \begin{Bmatrix} x\,, & \text{if } x \geq 0 \\ 0\,, & \text{if } x < 0 \end{Bmatrix}} \quad \textbf{(3.10)}$$

Cette fonction rectifie les valeurs des entrées inférieures à zéro en les forçant à être nulles, ce qui permet d'éliminer le problème de gradient qui s'annule, observé avec les premiers types de fonctions d'activation. La fonction ReLU a été utilisée dans les unités cachées des réseaux neuronaux profonds, associée à une autre fonction d'activation utilisée dans les couches de sortie du réseau. Des exemples typiques de cette utilisation se retrouvent dans les applications de classification d'objets et de reconnaissance vocale **[51]**.

La ReLU et ses variantes ont été utilisées dans différentes architectures d'apprentissage profond, restreintes de Boltzmann Machines, et les architectures de réseaux neuronaux convolutifs, bien que (Nair et Hinton, 2010) aient souligné que la fonction ReLU a été utilisée dans de nombreuses architectures en raison de sa simplicité et de sa fiabilité **[49]**.

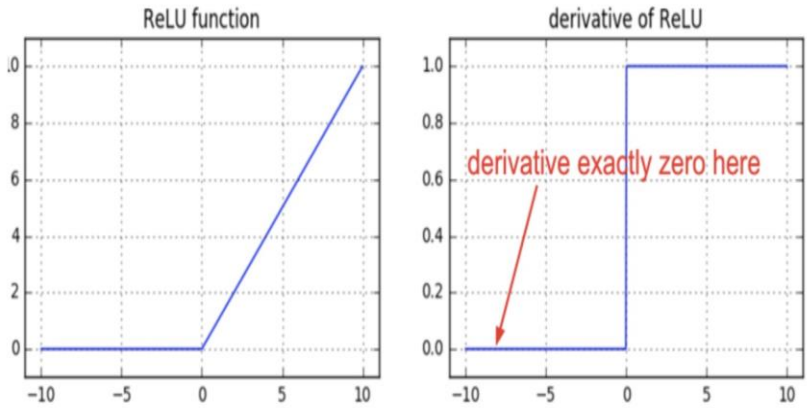


Figure 3.11: Fonction d'activation ReLU (Rectified Linear Unit).

- **Les caractéristiques [45]**

- Simple et rapide à calculer.
- Aide à résoudre le problème du (**vanishing gradient**) en laissant passer les gradients pour les valeurs positives.
- Très populaire dans les réseaux de neurones modernes.
- C'est la fonction d'activation la plus utilisée.
- Il est facile à calculer pour que le réseau de neurones converge très rapidement.
- Comme sa dérivée n'est pas 0 pour les valeurs positives du neurone

($f'(x) = 1\ pour\ x \geq 0$), ReLu ne sature pas et aucun neurone mort n'est rapporté. La saturation et le gradient de fuite ne se produisent que pour les valeurs négatives qui, données à ReLu, sont transformées en 0.

- Ce n'est pas une fonction centrée sur zéro.

- **Avantages**

- Performances généralement meilleures que les fonctions sigmoïdes ou tangentes hyperboliques.
- Moins coûteuse en termes de calculs.
- Le principal avantage de l'utilisation des unités linéaires rectifiées (ReLU) en calcul est qu'elles garantissent un calcul plus rapide, car elles n'effectuent pas d'exponentiations ni de divisions, ce qui améliore la vitesse globale de calcul.

- **Inconvénients**

- Peut entraîner le phénomène de **neurones morts** si trop de neurones ont des activations nulles.

- **Problème avec ReLU [51][48]**

ReLU fonctionne très bien dans la plupart des applications, mais ce n'est pas parfait. Il souffre d'un problème connu sous le nom de ReLU mourant (Mourir ReLU).

Pendant l'entraînement, certains neurones meurent effectivement, ce qui signifie qu'ils cessent de produire autre chose que 0. Dans certains cas, vous pouvez constater que la moitié des neurones de votre réseau sont morts, surtout si vous avez utilisé un taux d'apprentissage élevé. Un neurone meurt lorsque ses poids sont modifiés de telle sorte que la somme pondérée de ses entrées soit négative pour toutes les instances de l'ensemble d'apprentissage. Lorsque cela se produit, il continue à produire des 0, et la descente du gradient ne l'affecte plus puisque le gradient de la fonction ReLU est 0 lorsque son entrée est négative.

Pour résoudre ce problème des neurones morts, la ReLU modifiée, connue sous le nom de **Leaky ReLU**, a été proposée.

3.4.10 Leaky ReLU

La Leakey Relu permet d'ajouter une variante pour les nombres négatifs, ainsi les neurones ne meurent jamais. Ils entrent dans un long coma mais on toujours la chance de se réveiller à un moment donné. Intervalle de sortie$(\infty, +\infty)$ (**voir Figure 3.12**).

La Leaky ReLU, proposée en 2013, est une fonction d'activation qui introduit une petite pente négative à la ReLU afin de maintenir et préserver les mises à jour de poids actives tout au long du processus de propagation. Le paramètre alpha a été introduit comme solution aux problèmes de neurones morts associés à la ReLU, de sorte que les gradients ne soient jamais nuls à aucun moment pendant l'entraînement. La LReLU calcule le gradient avec une très petite valeur constante pour le gradient négatif, généralement de l'ordre de **0,01**. Ainsi, la fonction d'activation LReLU est calculée comme suit **[52]** :

$$\mathbf{f(x) = max(0, \alpha x) = \begin{cases} x & \text{if } x \geq 0 \\ \alpha x & \text{if } x < 0 \end{cases}} \quad \textbf{(3.11)}$$

Ou:

- $\boldsymbol{\alpha}$ est cte égale 0.01.

$$\mathbf{\acute{f}(x) = \begin{cases} 1 & \text{if } x \geq 0 \\ \alpha & \text{if } x < 0 \end{cases}} \quad \textbf{(3.12)}$$

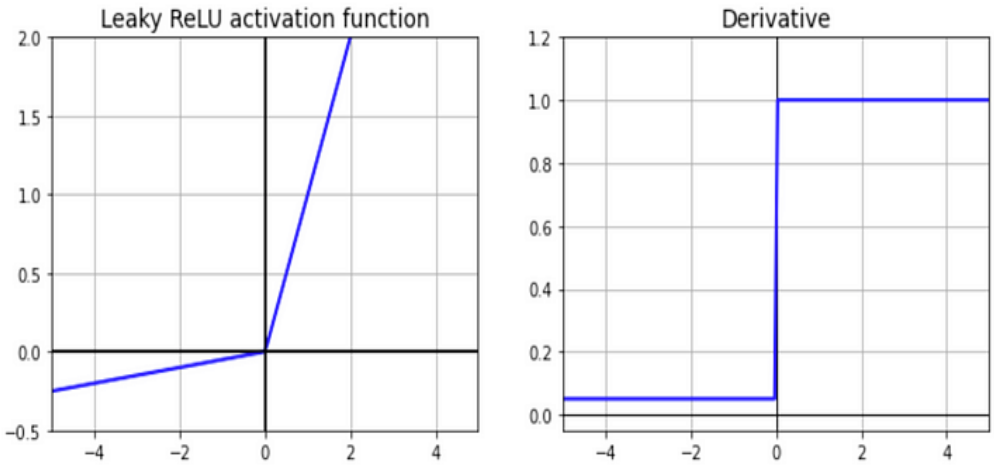


Figure 3.12: Fonction d'activation Leakey Relu et la derivé.

- **Caractéristiques**

- Une variante améliorée de ReLU qui permet un gradient non nul même pour les valeurs négatives. **[45]**

- Réduit le risque de (neurones morts). **[45]**

- **Avantages**

- Évite le problème des neurones morts en introduisant une pente faible pour les valeurs négatives.

- **Inconvénients**

- Plus complexe que ReLU, mais reste relativement simple.

3.4.11 ELU (Exponential Linear Unit)

Autre dérivé de la ReLU. Celle-ci va approcher les valeurs moyenne proche de 0, ce qui va avoir comme impact d'améliorer les performances d'entrainements. Elle utilise exponentiel pour la partie négative et non plus une fonction linéaire. Elle parait plus performante en expérimentation que les autres Relu. Pas de soucis de neurone mort (dying ReLU). Intervalle de sortie $(-\infty, +\infty)$. Les unités linéaires exponentielles (Exponential Linear Units, ELUs) sont un autre type de fonction d'activation (**voir Figure 3.13**), proposées par Clevert et al., en 2015. Elles sont utilisées pour accélérer l'entraînement des réseaux neuronaux profonds. L'avantage principal des ELUs réside dans leur capacité à atténuer le problème du gradient qui s'annule, en utilisant une identité pour les valeurs positives, tout en améliorant les caractéristiques d'apprentissage. Elles génèrent des valeurs négatives, ce qui permet de rapprocher l'activation moyenne des unités de zéro, réduisant ainsi la complexité computationnelle et améliorant la vitesse d'apprentissage **[57]**.

Les ELUs constituent une bonne alternative à la ReLU, car elles réduisent les biais de décalage en poussant l'activation moyenne vers zéro pendant l'entraînement. L'unité linéaire exponentielle (ELU) est définie par :

$$\mathbf{f(x)} = \begin{pmatrix} \mathbf{x}\,, & \mathbf{if\ x > 0} \\ \boldsymbol{\alpha}.\,\mathbf{e^x - 1}, & \mathbf{if\ x \leq 0} \end{pmatrix} \quad \textbf{(3.13)}$$

La dérivée ou le gradient de l'équation ELU est donné par :

$$\mathbf{f'} = \begin{pmatrix} \mathbf{1}\,, & \mathbf{if\ x > 0} \\ \mathbf{f(x) + \alpha}\,, & \mathbf{if\ x \leq 0} \end{pmatrix} \quad \textbf{(3.14)}$$

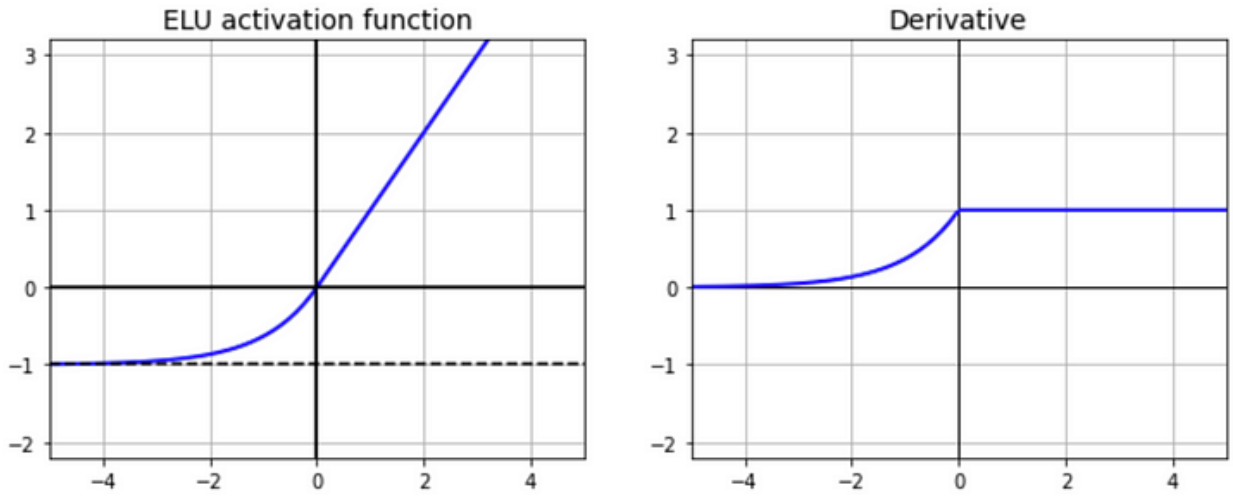


Figure 3.13 : Fonction d'activation ELU (Exponential Linear Unit) et la derivé.

Nous pouvons voir dans la figure **(3.11)** que :

- ELU a modifié la pente de la partie négative de la fonction.
- Contrairement aux fonctions Leaky ReLU et PReLU, au lieu d'une ligne droite, ELU utilise une courbe logarithmique pour les valeurs négatives **[58]**.

- **Problème avec ELU**

Le principal inconvénient de l'activation de l'ELU est qu'elle est plus lente à calculer que le ReLU et ses variantes (en raison de l'utilisation de la fonction exponentielle), mais pendant l'apprentissage, cela est compensé par le taux de convergence plus rapide. Cependant, au moment du test, un réseau ELU sera plus lent qu'un réseau ReLU **[57]**.

3.5 Aspects des functions d'activation

Tableau 3.1 : Les differents aspects des fonctions d'activation.

Aspect	**Description**
Initialisation des poids	- L'efficacité dépend de l'initialisation. - Orthogonale ou asymétrique (pour ReLU) permet une meilleure convergence et évite les neurones morts.
Adaptation aux types de données	- La plupart des fonctions sont testées sur des images , peu de travaux les étudient sur d'autres données comme le texte ou les séries temporelles.
Complexité du réseau et approximation	- Les fonctions comme ReLU permettent une meilleure approximation dans les réseaux profonds ; un seul MLP ReLU peut atteindre un optimum global.
Surconfiance et incertitude	- Certaines AFs produisent des prédictions trop sûres loin des données, cela peut être corrigé par apprentissage avec confiance adversariale.
Valeurs singulières et régularisation	- Les valeurs singulières des couches ReLU affectent l'apprentissage. - Une approximation gaussienne peut corriger la surconfiance du modèle.

3.6 Problèmes récurrents des fonctions d'activation

Les fonctions standards amènent au réseau la disparition ou l'explosion de gradient.

Voici une liste d'éventuels problèmes que vous pouvez rencontrer concernant ce chapitre :

3.6.1 Disparition du gradient (Vanishing Gradient)

- Survient avec des fonctions comme Sigmoid et Tanh.
- Lorsque l'algorithme progresse vers les couches inférieures du réseau, les gradients deviennent de plus en plus petits.

- **Résultat** : les poids des connexions de ces couches sont à peine mis à jour → apprentissage très lent, voire bloqué.

3.6.2 Explosion du gradient (Exploding Gradient)

- Inverse du problème précédent : les gradients deviennent trop grands.
- Les poids augmentent rapidement et peuvent rendre l'apprentissage instable ou divergent.

3.6.3 Neurones morts (Dying ReLU)

- Spécifique à la fonction ReLU.
- Lorsqu'un neurone reçoit toujours une entrée négative, sa sortie est bloquée à zéro.
- Ce neurone cesse alors d'apprendre (gradient nul). Cela peut affecter un grand nombre de neurones dans les réseaux profonds.

3.6.4 Saturation

- Les fonctions comme Sigmoid et Tanh saturent lorsque l'entrée est trop grande ou trop petite.
- La dérivée devient presque nulle **c-à-d** pas de mise à jour de poids **c-à-d** apprentissage inefficace.
- Cela rend la rétropropagation difficile.

3.6.5 Calcul coûteux:

- Certaines fonctions comme Sigmoid, Tanh, Softplus ou Mish nécessitent des calculs exponentiels, ralentissant l'entraînement.
- Ce facteur est particulièrement problématique sur de grands jeux de données ou dans les architectures profondes.

3.6.6 Sortie non centrée sur zero

- Par exemple, Sigmoid produit une sortie entre 0 et 1.
- Cela peut provoquer un déséquilibre dans les mises à jour des poids, ralentissant l'optimisation.

Tableau 3.2: Problèmes des fonctions d'activation.

Problème	**Description**	**Fonctions concernées**
Disparition du gradient	Les dérivées deviennent très petites → pas de mise à jour des poids	Sigmoid, Tanh, Softsign, Softplus
Explosion du gradient	Les dérivées deviennent très grandes → divergence de l'apprentissage	Peut affecter toutes si mauvaise init. /taux
Neurones morts (Dying ReLU)	Neurones bloqués à zéro à cause d'entrées toujours négatives	ReLU
Saturation	Fonction 'bloquée' pour valeurs extrêmes, dérivée ≈ 0	Sigmoid, Tanh
Calcul coûteux	Temps de calcul élevé dû à des opérations complexes (exponentielles, etc.)	Sigmoid, Tanh, Softplus, Mish, GELU
Sortie non centrée sur zéro	Peut ralentir l'apprentissage (poids mis à jour dans une seule direction)	Sigmoid, ReLU
Sensibilité aux hyperparamètres	Nécessite un réglage précis du taux d'apprentissage, etc.	Mish, GELU, ELU

Tableau 3.3: Les aventages et inconvénientes des fonctions d'activation.

Fonction d'activation	Avantages	Inconvénients
ReLU	- Simple, rapide à calculer. - Pas de vanishing gradient pour x>0	- Neurones 'morts' pour x<0 (gradient nul)
Leaky ReLU	- Évite le problème des neurones morts. - Gradient non nul pour x<0	- Plus complexe que ReLU, performance similaire
ELU	- Évite le gradient nul. - Bon pour les réseaux profonds	- Plus lent que ReLU (calcul exponentiel)
Swish	- Lisse, non monotone. - Bonnes performances générales. - Pas de gradient nul.	- Plus, coûteux que ReLU.
Mish	- Très bonnes performances - Conserve les valeurs négatives - Stabilité du gradient	- Plus lent à calculer Moins répandue
GELU	- Performances optimales - Activation douce - Utilisé dans des architectures avancées	- Coût computationnel élevé Sensible aux hyperparamètres
Sigmoid	- Sortie entre 0 et 1 (utile pour classification binaire)	- Vanishing gradient Saturation - Non centré sur zéro - Coût de calcul élevé
Tanh	- Sortie centrée sur 0 - Meilleur que sigmoid pour les couches cachées	Vanishing gradient Saturation Calcul coûteux
Softsign	- Transition douce - Sortie bornée entre -1 et 1	- Diminution rapide pour grandes valeurs - Peu utilisé
Softplus	- Lisse et différentiable - Pas de sortie nulle	- Calcul lent - Moins performant pour valeurs négatives
Softmax	- Fournit une distribution de probabilité (multi-classe)	- Seulement pour les couches de sortie

CHAPITRE 3 : LES DIFFERENTES FONCTIONS D'ACTIVATION DES RÉSEAUX NEURONES ET LEUR IMPACT SUR LES PERFORMANCES DE DIVERS MODÈLES CNN POUR LA CLASSIFICATION

3.7 Choix de la Fonction d'Activation

Le choix de la fonction d'activation dépend de l'emplacement dans le réseau (couche cachée ou de sortie) et du type de tâche (classification binaire, multi-classe ou régression).

- **Pour les couches cachées**

ReLU est la fonction la plus couramment utilisée, suivie de sa variante (Leaky ReLU).

- **Pour les couches de sortie**

- Utiliser (**sigmoid**) pour les problèmes de classification binaire.
- Utiliser (**softmax**) pour les problèmes de classification multi-classe.
- Utiliser (**linear**) pour les problèmes de régression.

- **Tableau 3.4:** Le choix de la fonction d'activation.

Position dans le réseau	**Fonction recommandée**	**Justification**
Couches caches	ReLU	Simple, rapide, efficace et largement utilisée.
Couches cachées	Leaky ReLU, PReLU, ELU	Alternatives à ReLU pour éviter les neurones morts.
Couches cachées	Mish, Swish	Bonnes performances, mais plus coûteuses et moins répandues.
Couche de sortie (binaire)	Sigmoid	Fournit une probabilité entre 0 et 1, utile pour la classification binaire.
Couche de sortie (multi-classe)	Softmax	Fournit une distribution de probabilités sur plusieurs classes (somme = 1).
Couche de sortie (régression)	Linéaire (aucune activation)	Utilisée pour produire des valeurs continues sans limitation de plage.

CHAPITRE 3 : LES DIFFERENTES FONCTIONS D'ACTIVATION DES RÉSEAUX NEURONES ET LEUR IMPACT SUR LES PERFORMANCES DE DIVERS MODÈLES CNN POUR LA CLASSIFICATION

Tableau 3.5: Tableau comparatif des fonctions d'activation.

Fonction	**Intervalle de sortie**	**Caractéristiques principales**	**Avantages**	**Inconvénients**
Sigmoid	(0, 1)	Non-linéaire, saturable	Donne une probabilité, utile en sortie pour classification	Vanishing gradient, non centrée sur 0, calcul coûteux
Tanh	(-1, 1)	Centrée sur 0, plus lisse que Sigmoid	Meilleure que Sigmoid, facilite backpropagation	Toujours vanishing gradient, calcul coûteux
ReLU	(0, ∞)	Simple, rapide à calculer	Évite vanishing gradient, très populaire	Neurones morts (Dying ReLU), non centrée sur 0
Leaky ReLU	(-∞, ∞)	Variante ReLU, pente légère pour valeurs négatives	Évite les neurones morts	Moins simple que ReLU, hyperparamètre α
ELU	(-α, ∞)	Fonction exponentielle pour $x < 0$	Réduction du biais de sortie, bon gradient	Calcul plus lent, dépendant d'un hyperparamètre
Softplus	(0, ∞)	Version lissée de ReLU	Différentiable, pas de zéro strict	Calcul plus coûteux, moins efficace sur valeurs négatives
Softsign	(-1, 1)	Lissage du gradient	Transitions douces, sortie bornée	Lent pour valeurs extrêmes, peu utilisé
GELU	(-∞, ∞)	Activation probabiliste, utilisée dans Transformers	Bonne convergence, différentiable, conserve valeurs négatives	Calcul complexe, dépend de Φ(x)
Swish	(-∞, ∞)	Non-monotone, douce	Améliore la généralisation, bon gradient	Plus coûteuse que ReLU
Mish	(-∞, ∞)	Lisse, non monotone	Très bonnes performances, auto-régularisante	Calcul complexe, encore peu utilisé
Softmax	(0, 1), somme = 1	Utilisée en sortie multi-classe	Distribution de probabilité	Non utilisée dans les couches cachées

3.8 Conclusion :

En résumé, le choix de la fonction d'activation est essential pour la performance et l'apprentissage des réseaux de neurones. Des fonctions simples comme ReLU ont changeé l'apprentissage profond grâce à leur efficacité, mais leurs limitations ont mené à l'élaboration de variantes telles que Leaky ReLU, ELU, Swish ou Mish, qui visent à corriger les problèmes de gradients nuls ou saturés. Pour les couches de sortie, sigmoid et softmax restent les références, selon qu'il s'agisse de classification binaire ou multi-classe. Ce chapitre illustre qu'aucune fonction n'est universellement supérieure : leur choix doit être adapté au type de tâche, à la structure du réseau, et aux données traitées.

CHAPITRE 4 : L'IMPLÉMENTATION SUR MATLAB

CHAPITRE 4 : L'IMPLIMENTATION SUR MATLAB

4.1 Introduction

Dans ce chapitre, nous explorons les différents outils et approches méthodologiques utilisés dans la conception d'un réseau de neurones appliqué à la reconnaissance d'images. Nous débutons par la présentation de MATLAB, expliqué non seulement comme un langage de programmation, mais aussi comme un environnement complet permettant de réaliser des calculs numériques complexes, de manipuler des matrices, de tracer des fonctions et de visualiser des données. Ensuite, l'utilisation de l'optimisateur ADAM est détaillée : ce dernier, méthode issue de la descente de gradient, est configuré à l'aide de paramètres précis pour permettre un apprentissage rapide et efficace, comme illustré par l'exemple de configuration dans le code. Par la suite, le chapitre passe en revue plusieurs bases de données emblématiques dans le domaine de la vision par ordinateur, telles que MNIST, SVHN et Fashion-MNIST. Pour chacune de ces bases, une analyse comparative des fonctions d'activation (Relu, LeakyReLU, Swish, Mish, GELU, etc.) est présentée à travers différentes métriques de performance – Overall Accuracy (OA), Average Accuracy (AA) et Kappa – et illustrée par des graphiques d'apprentissage et de validation. L'ensemble de ces remarques et figures vise à démontrer comment le choix de la fonction d'activation impacte directement la performance, la convergence et la capacité de généralisation des modèles de deep learning.

4.2 Logiciels utilisés : MATLAB

4.2.1 Définition

MATLAB est à la fois un langage de programmation et un environnement qui offre des fonctionnalités avancées pour effectuer des calculs numériques. Il permet d'accomplir diverses tâches, notamment la manipulation de matrices, le tracé de fonctions, la visualisation de données, le traitement d'images, l'implémentation de nouveaux algorithmes et la création d'interfaces graphiques. [2]

4.3 Optimisateur utiliser : ADAM

Nous avons fait le choix d'utiliser empiriquement l'optimsateur Adam (Un type de Descente de Gradient adatatif decrit precedement dans le chapitre deux) parmis d'autres optimisateur tel que Sgdm.

Pour cette optimizateur on à prix 10^{-3} comme valeur intiale de tau d'apprentissage, un nombre d'epoques de 5, une taille de minilot de 128 avec un mélange de données à chaque époque.

CHAPITRE 4 : L'IMPLIMENTATION SUR MATLAB

4.4 Partie d'application

4.4.1 MNIST

Notre base des données **MNIST** pour **M**odified ou **M**ixed **N**ational **I**nstitute of **S**tandards and **T**echnology la figure (4.1), est une base de données de chiffres écrits à la main. Ce sont des images en noir et blanc, normalisées centrées de 28 x 28 pixels elle est constituer d'un lot de 70000 images de diffèrent chiffre de 0 à 9. Cette basse est organier sous forme de 10 lot contenant 7000 images représentant différente image d'un seul chiffre dont voici

Figure 4.1: Echantillon de la base MNIST

- **Rappel de la définition de la Fonction D'activation**

Une fonction d'activation est une fonction mathématique inspirée du fonctionnement des neurones biologiques. Elle détermine si un neurone artificiel doit s'activer ou non, en fonction du signal reçu. Autrement dit, elle permet ou bloque le passage de l'information selon un certain seuil.

Tableau 4.1: Valeur des metriques OA, AA, et Kappa pour differentes FA appliqué à la base des données Minist.

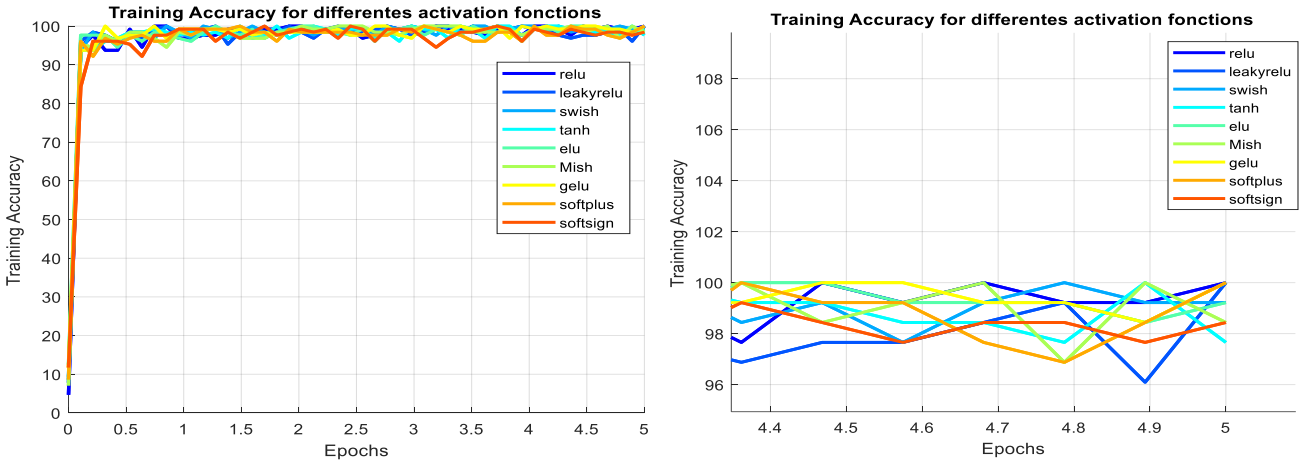
Fonction d'activation (FA)	Relu	Leakyrelu	Swish	Tanh	Elu	Mish	Gelu	Softplus	Softsign
OA%	98,73	99,21	99,21	98,92	99,02	99,08	98,97	99,03	98,49
AA%	98,76	99,21	99,21	98,92	99,02	99,07	98,95	99,04	98,49
Kappa x 100	98,59	99,12	99,12	98,80	98,91	98,98	98,86	98,92	98,32

Le tableau (4.1) présente les performances de plusieurs fonctions d'activation selon trois métriques : OA% (Overall Accuracy)**,** AA% (Average Accuracy) et Kappa

- **Observations sur le Tableau 4.1**

- **Meilleures performances :** Les fonctions **Swish** et **LeakyReLU** obtiennent les meilleures performances globales avec un OA = 99.21%, AA = 99,21% et kappa x 100 = 99.12
- **Fonctions les moins performantes :** Softsign affiche les résultats les plus faibles avec environ **98,49% (OA)**, ce qui reste toutefois une performance élevée.

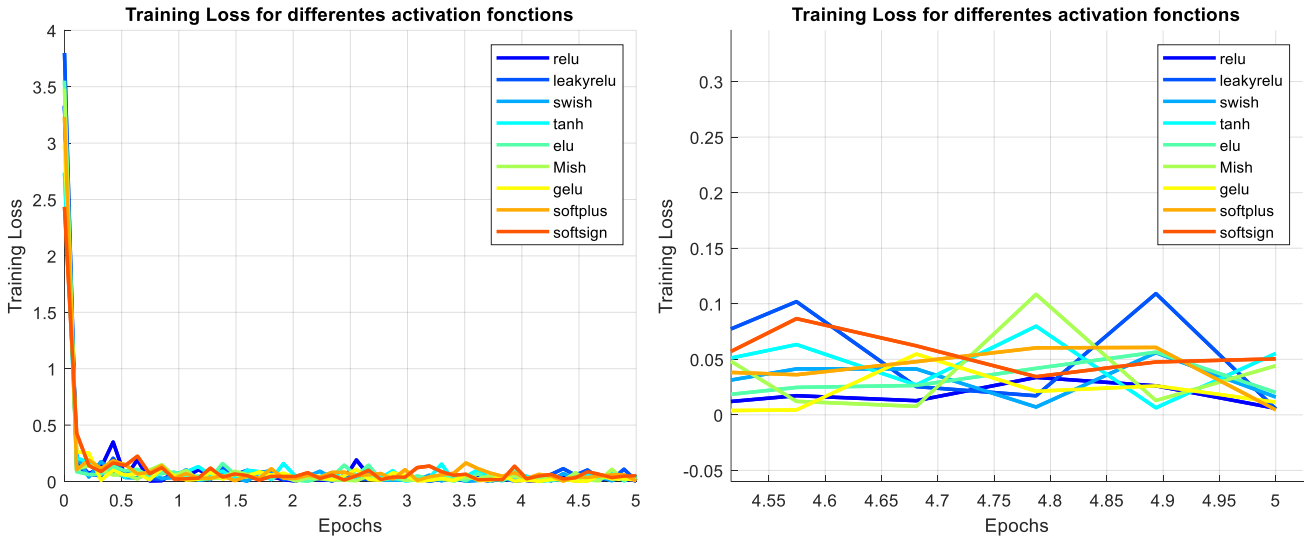
On remarque : Les différences sont minimes (moins de 1% d'écart entre la meilleure et la moins bonne FA), ce qui montre que toutes les fonctions d'activation sont relativement efficaces sur MNIST.



(a) (b)

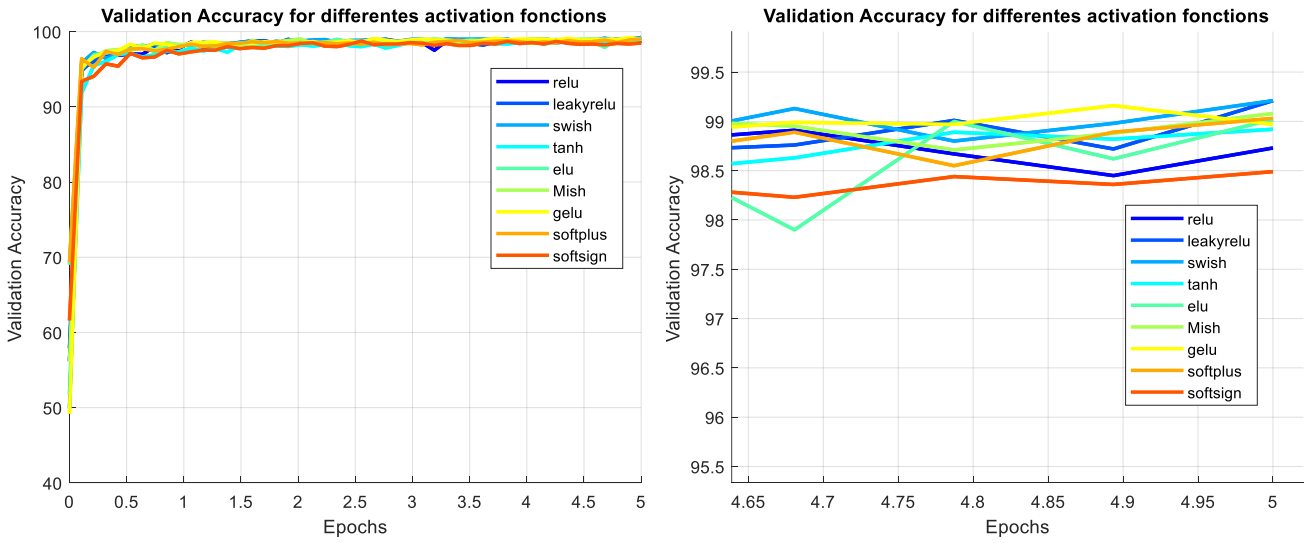
Figure 4.2: (a) Précision d'apprentissage pour differente FA pour la base des données Minist, (b) Zoom de (a)

CHAPITRE 4 : L'IMPLIMENTATION SUR MATLAB



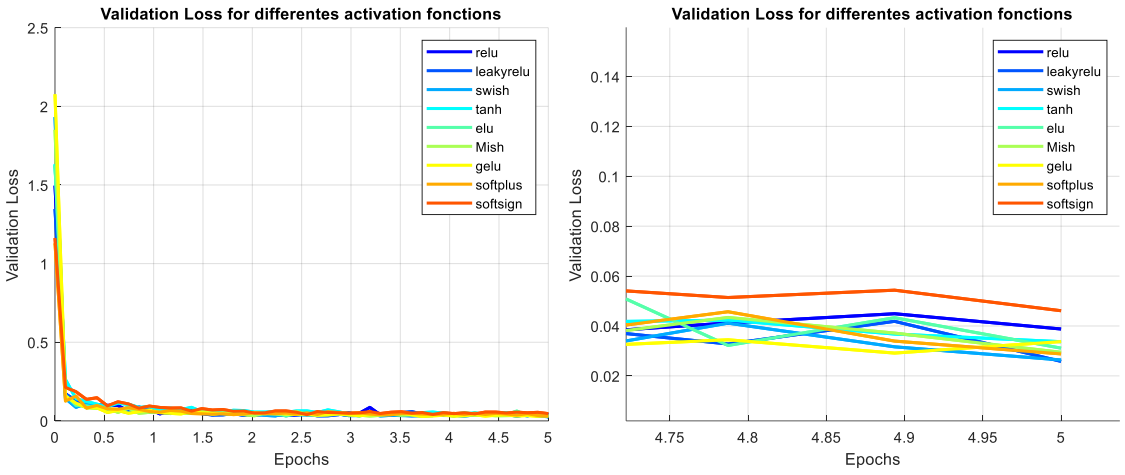
(c) (d)

Figure 4.3: (c) Erreur (perte) d'apprentissage pour differente FA pour la base des données Minist, (d) Zoom de (c)



(e) (f)

Figure 4.4: (e) Precission de validation pour differente FA pour la base des données Minist, (f) Zoom de (e)



(g) (h)

Figure 4.5: (g) Erreur (perte) de validation pour differente FA pour la base des données

- **Observations générales (figures)**

Les figures montrent les courbes d'apprentissage et de validation pour chaque FA :

- **Figure 4.2 (a, b) – Précision d'apprentissage**
- Les courbes sont globalement ascendantes.
- Swish, LeakyReLU, ELU, Mish présentent une convergence vers une stablitée et une haute précision.
- Zoom (b) montre des différences plus nettes en fin d'apprentissage.
- **Figure 4.3 (c, d) – Erreur d'apprentissage**
- Tendance décroissante pour toutes les FA.
- Les fonctions les plus performantes (Swish, Mish) descendent et atteignent des erreurs minimales.
- **Figure 4.4 (e, f) – Précision de validation**
- Similaire à la précision d'apprentissage.
- La stabilité des courbes indique une bonne généralisation pour les meilleures FA.
- **Figure 4.5 (g, h) – Erreur de validation**
- Les fonctions les plus performantes gardent une perte faible en validation

- **Conclusion (MNIST)**
- Les fonctions Swish et LeakyReLU offrent les meilleurs compromis entre performance, stabilité de convergence.
- Toutes les fonctions d'activation testées restent efficaces sur MNIST, mais certaines comme Softsign et Tanh montrent une légère infériorité.
- L'analyse graphique complète bien les résultats numériques en illustrant la qualité de l'apprentissage et la capacité de généralisation.

4.4.2 The Street View House Numbers (SVHN) Dataset

SVHN est un ensemble de données d'images du monde réel conçu pour le développement d'algorithmes d'apprentissage automatique et de reconnaissance d'objets, avec un besoin minimal de prétraitement et de formatage des données. Il peut être considéré comme similaire à MNIST (par exemple, les images sont des petits chiffres recadrés), mais il intègre un ordre de grandeur supérieur de données annotées (plus de 600 000 images de chiffres) et provient d'un problème réel nettement plus complexe (la reconnaissance des chiffres et des nombres dans des images de scènes naturelles). SVHN est obtenu à partir des numéros de maison dans les images de Google Street View.

Aperçu

- 10 classes, une pour chaque chiffre. Le chiffre '1' a l'étiquette 1, '9' a l'étiquette 9 et '0' a l'étiquette 10.
- 73 257 chiffres pour l'entraînement, 26 032 chiffres pour les tests, et 531 131 échantillons supplémentaires, légèrement moins difficiles, à utiliser comme données d'entraînement supplémentaires.
- Existe en deux formats:

- Images originales avec des boîtes de délimitation au niveau des caractères.
- Images de 32x32 pixels, similaires à MNIST, centrées autour d'un seul caractère (beaucoup de ces images contiennent des éléments perturbateurs sur les côtés).

CHAPITRE 4 : L'IMPLIMENTATION SUR MATLAB

Figure 4.6: Numéros complets

Tableau 4.1: Valeur des metriques OA, AA, et Kappa pour differentes FA appliqué à la base des données Minist.

FA	Relu	leakyrelu	swish	tanh	elu	Mish	gelu	softplus	Softsign
OA %	90.85	91.23	91.47	89.60	89.47	91.72	92.01	91.12	89.03
AA %	90.23	90.68	91.05	88.96	88.39	91.15	91.66	90.09	88.41
Kappa x 100	89.61	90.05	90.32	88.20	88.06	90.61	90.93	89.94	87.54

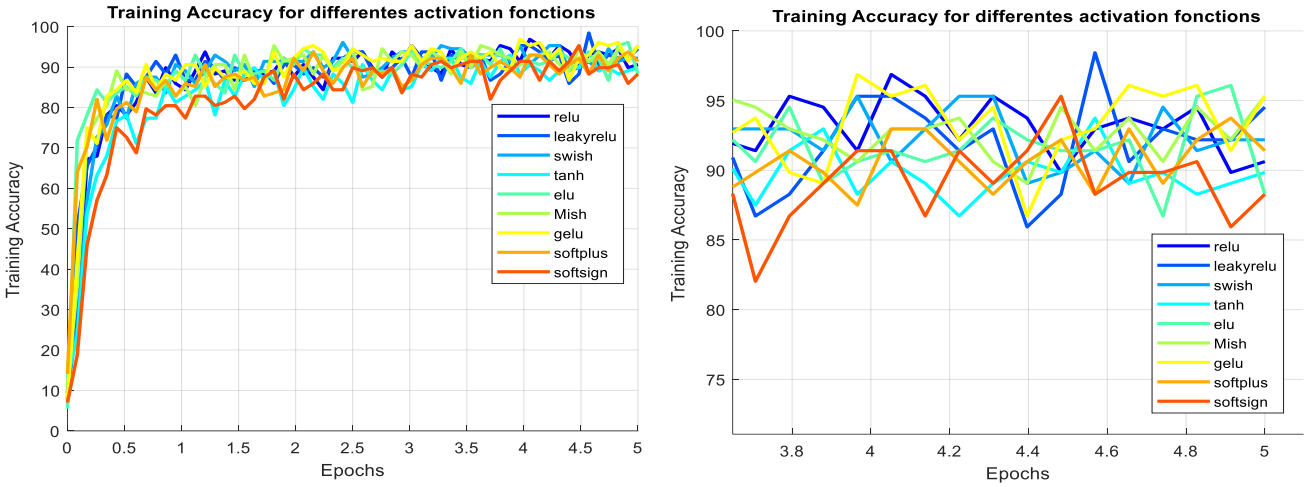
Le tableau (4.2) présente les performances de plusieurs fonctions d'activation selon trois métriques : OA% (Overall Accuracy)**,** AA% (Average Accuracy) et Kappa.

- **Observations sur le Tableau 4.2**

Les meilleures performances sont obtenues par Gelu, suivie de Mish et Swish, selon les trois métriques : OA (Overall Accuracy), AA (Average Accuracy), et Kappa.

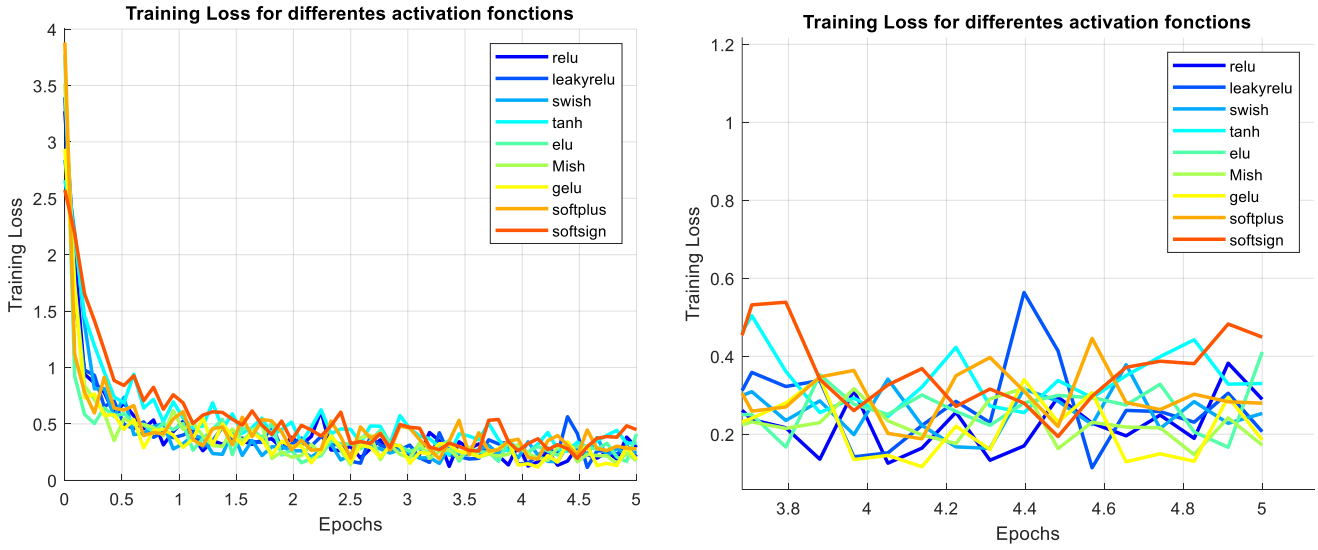
Gelu atteint les valeurs les plus élevées avec : **OA** : 92.01 %, **AA** : 91.66 %, **Kappa** : 90.9.

Tanh, Elu et surtout Softsign ont des résultats nettement inférieurs, indiquant une efficacité moindre sur la base de données SVHN.



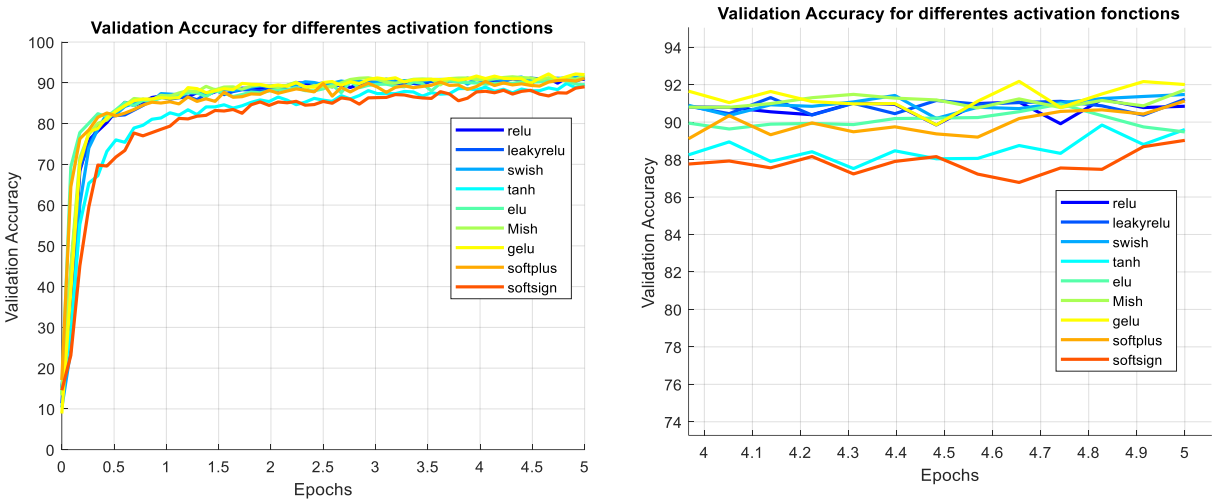
(a) (b)

Figure 4.7: (a) Précision d'apprentissage pour differente FA pour la base des données SVHN, (b) Zoom de (a)



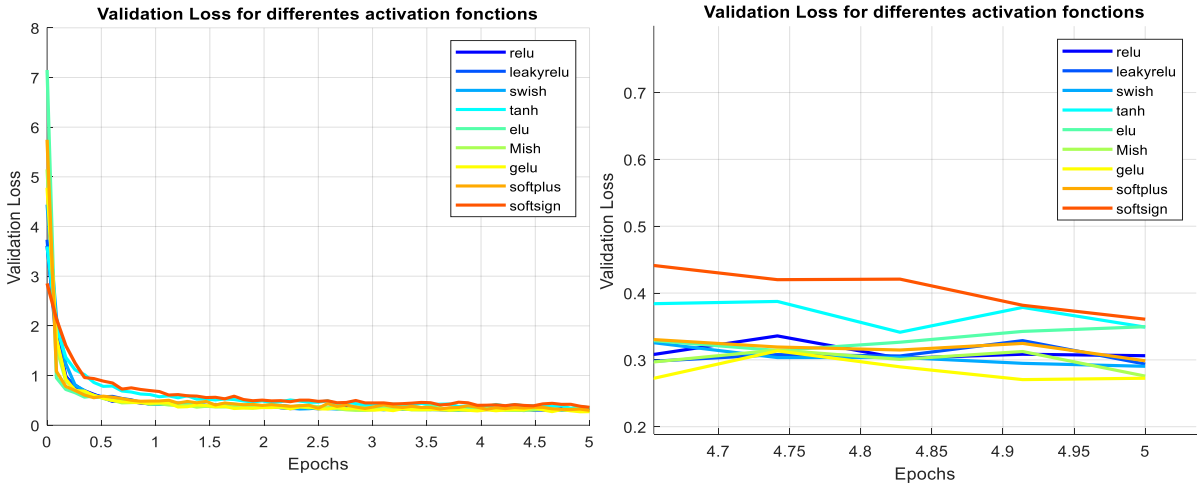
(c) (d)

Figure 4.8: (c) Erreur (perte) d'apprentissage pour differente FA, pour la base des données SVHN, (d) Zoom de (c)



(e) (f)

Figure 4.9: (e) Précisions de validation pour differente FA pour la base des données SVHN (f) Zoom de (e)



(g) (h)

Figure 4.10: (g) Erreur (perte) de validation pour différente FA pour la base des données SVHN(h) Zoom de (g)

- **Observations sur les Figures 4.7 à 4.10**

- **Figure 4.7 – Erreur (perte) d'apprentissage**

Les fonctions comme Gelu, Mish et Swish montrent une courbe de perte plus basse et stable, indiquant un apprentissage efficace. Les pertes pour Tanh, Elu et Softsign sont plus élevées et parfois moins stables.

- **Figure 4.8 – Précision d'apprentissage**

Gelu et Mish atteignent une précision d'apprentissage plus élevée que les autres fonctions d'activation. Relu et Swish offrent de bonnes performances, bien que légèrement inférieures à Gelu.

- **Figure 4.9 – Erreur de validation**

Les pertes de validation sont les plus faibles pour Gelu et Mish, montrant leur bonne capacité de généralisation. Les fonctions avec une plus forte perte (comme Softsign).

- **Figure 4.10 – Précision de validation**

La précision de validation confirme les résultats précédents : Gelu, Mish, et Swish conservent leur avantage. Tanh, Elu, et Softsign sont les moins performantes.

- **Conclusion (SVHN)**
- Les fonctions d'activation modernes telles que Gelu, Mish, et Swish surpassent les fonctions classiques comme Relu, Tanh, et Elu dans tous les aspects : apprentissage, validation, précision et stabilité.
- Gelu apparaît comme la fonction d'activation la plus performante pour la base de données SVHN.

4.4.3 Fashion-MNIST

Fashion-MNIST est un ensemble de données composé de 70 000 images en niveaux de gris de 28×28 pixels représentant des articles de mode répartis en 10 classes, avec 7 000 images par classe chaque exemple d'entraînement et de test est attribué à l'une des étiquettes. L'ensemble d'entraînement contient 60 000 images, tandis que l'ensemble de test en comprend 10 000. Fashion-MNIST partage la même taille d'image, le même format de données et la même structure de séparation entre l'entraînement et le test que le MNIST original.

- MNIST est trop facile. Les réseaux de neurones convolutionnels peuvent atteindre une précision de 99,7 % sur MNIST. Les algorithmes classiques d'apprentissage automatique peuvent aussi facilement obtenir 97 %.

- MNIST est surutilisé, chercheur chez Google Brain et expert en apprentissage profond, appelle à abandonner MNIST.

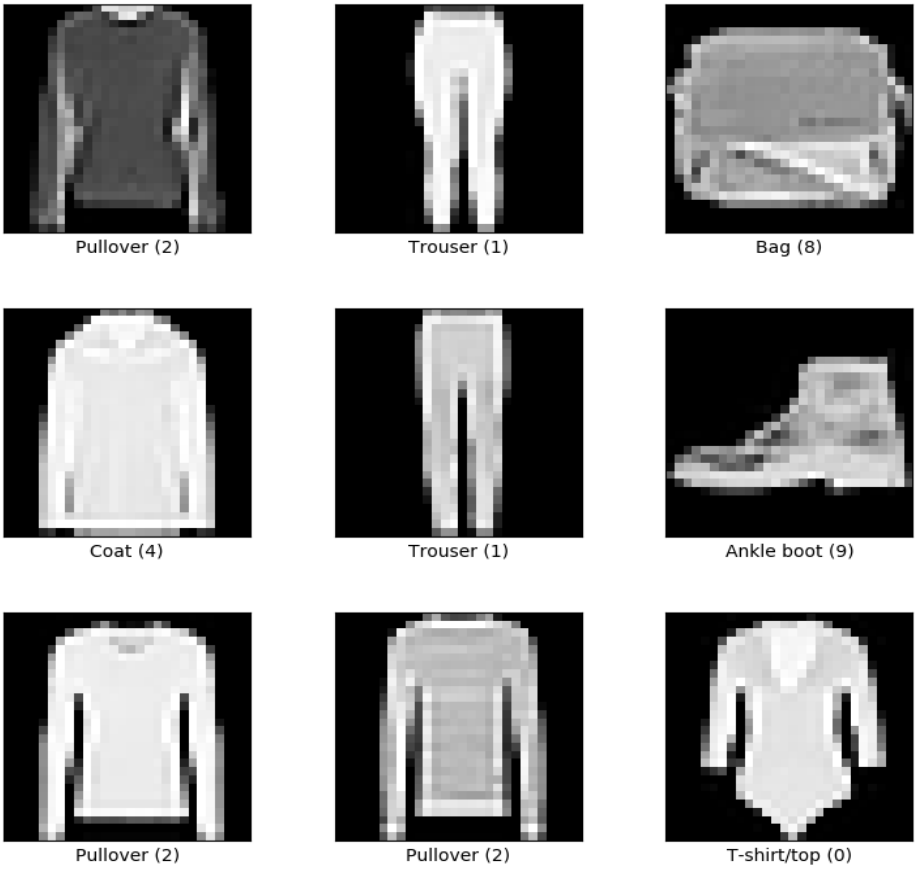


Figure 4.11: Les classes.

CHAPITRE 4 : L'IMPLIMENTATION SUR MATLAB

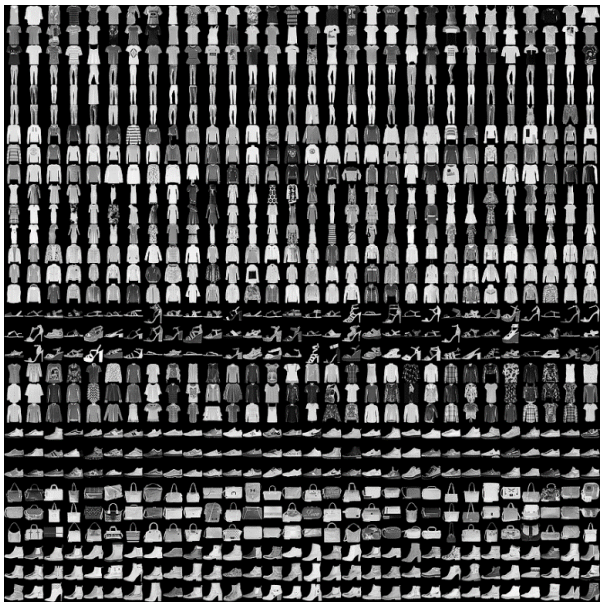


Figure 4.12: Example de l'apparence des données (Fashion Minist) (chaque classe occupe trois lignes)

Tableau 4.3: Valeur des metriques OA, AA, et Kappa pour differentes FA appliqué à la base de données Fashion-MNIST.

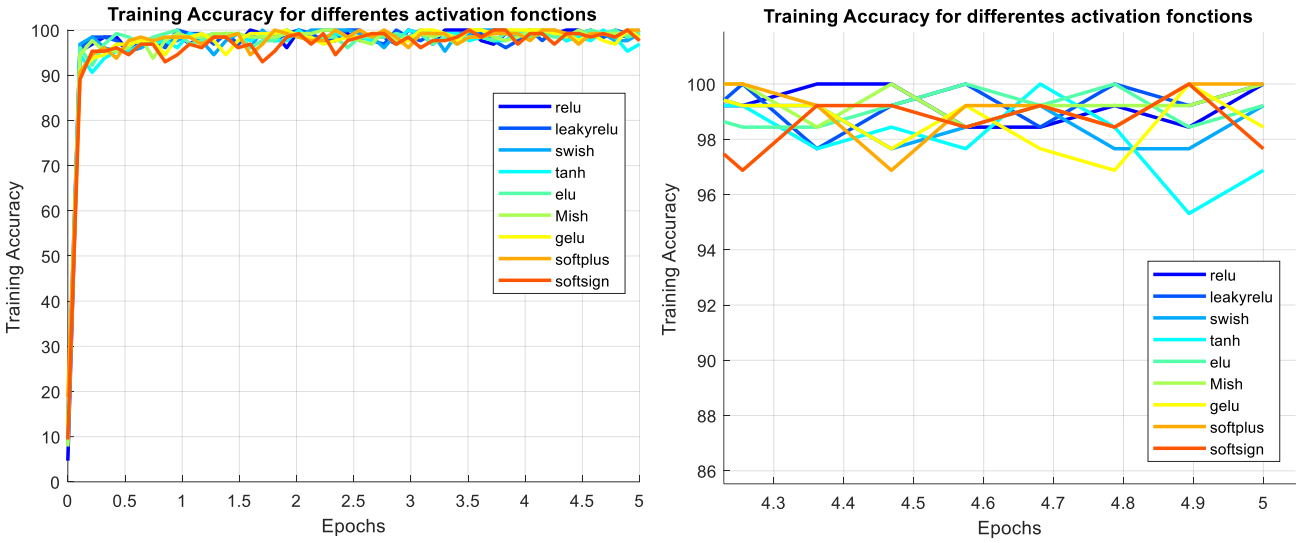
FA	Relu	leakyrelu	swish	tanh	elu	Mish	gelu	softplus	Softsign
OA%	99.11	98.90	98.96	98.62	98.74	99.12	99.05	98.80	98.84
AA%	99.10	98.91	98.97	98.62	98.72	99.12	99.04	98.79	98.83
Kappa x 100	99.01	98.78	98.84	98.47	98.60	99.02	98.94	98.67	98.71

- **Observations sur le Tableau 4.3:**

Toutes les fonctions d'activation présentent des performances très élevées, avec des valeurs de OA%, AA% et Kappa supérieures à 98 %.

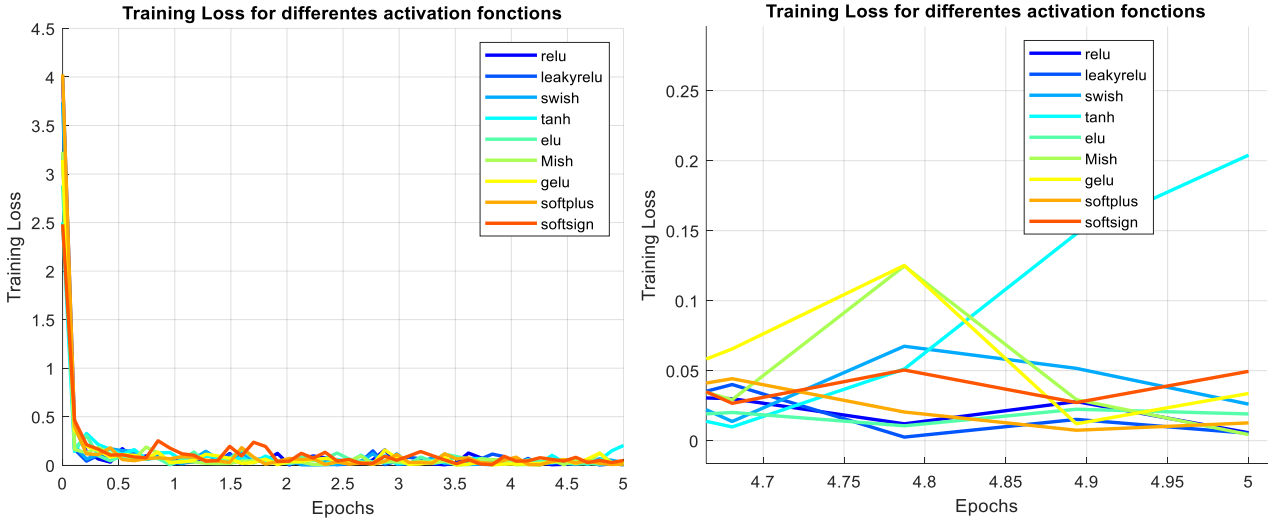
Mish obtient les meilleures valeurs pour les trois métriques : **OA** = 99.12%, **AA** = 99.12%**, Kappa** = 99.02.

Tanh présente les résultats les plus faibles parmi toutes les FA testées, bien qu'ils restent globalement bons.



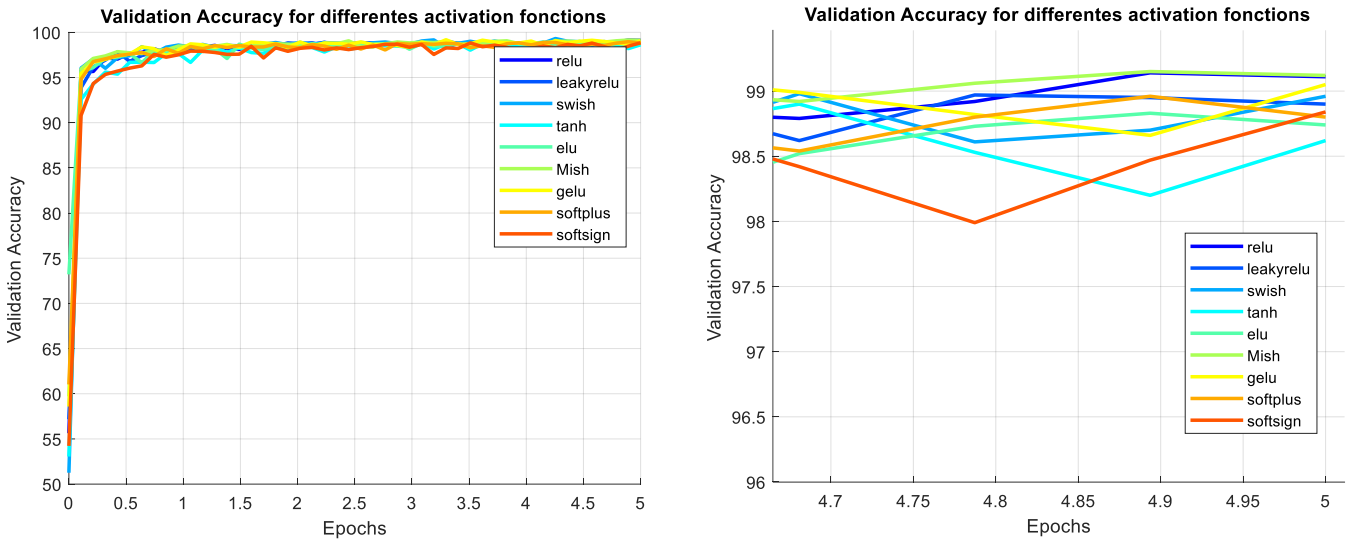
(a) (b)

Figure 4.13: (a) Précision d'apprentissage pour differente FA pour la base des données Fashion-MNIST, (b) Zoom de (a)



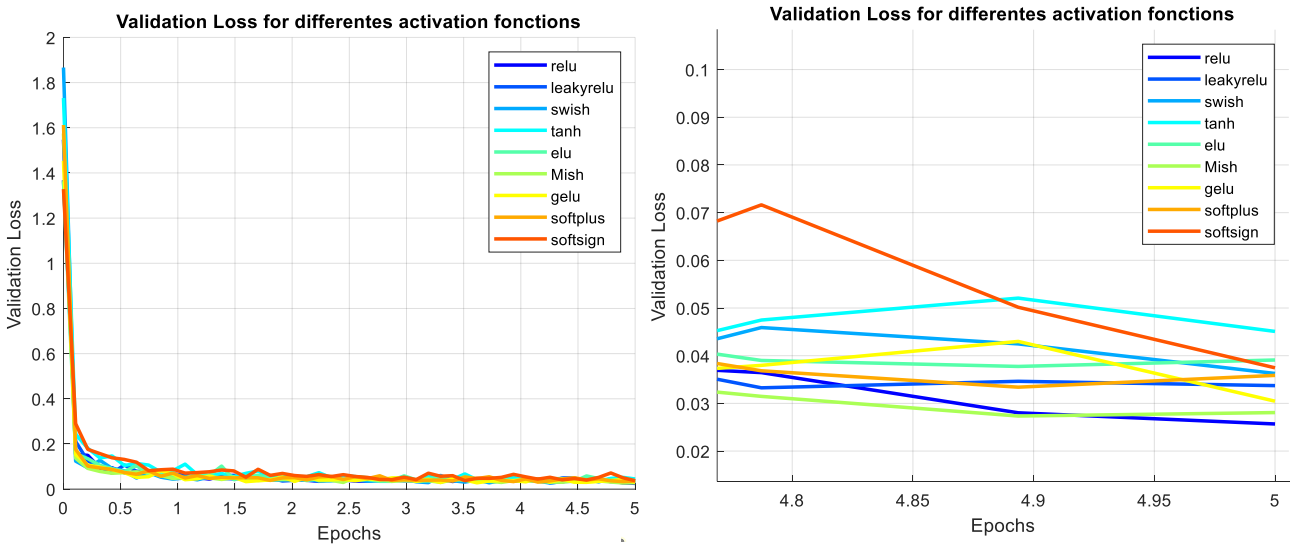
(c) (d)

Figure 4.14: (c) Erreur (perte) d'apprentissage pour differente FA, pour la base des données Fashion-MNIST, (d) Zoom de (c)



(e) (f)

Figure 4.15: (e) Précisions de validation pour differente FA pour la base des données Fashion-MNIST (f) Zoom de (e)



(g) (h)

Figure 4.16: (g) Erreur (perte) de validation pour différente FA pour la base des données Fashion-MNIST (h) Zoom de (g)

- **Observations sur les Figures 4.13 à 4.16**

- **Figure 4.13 – Précision d'apprentissage**

Mish, Relu, et Gelu montrent une montée rapide de la précision dès les premières époques.

Tanh et Softsign ont une montée plus lente, traduisant un apprentissage moins efficace.

- **Figure 4.14 – Erreur d'apprentissage**

Mish et Gelu obtiennent les pertes d'apprentissage les plus faibles, ce qui confirme leur bonne capacité de convergence.

Tanh et Softsign gardent des pertes plus élevées et plus Changeable.

- **Figure 4.15 – Précision de validation**

Les meilleures précisions de validation sont obtenues avec Mish, Relu et Gelu.

La stabilité de la précision en validation est un bon indicateur de la robustesse du modèle.

- **Figure 4.16 – Erreur de validation**

Les FA les plus performantes ont également les pentes de décroissance de perte les plus régulières et stables.

Les pertes sont plus élevées pour Tanh et Softsign, ce qui peut indiquer un risque de sous-apprentissage ou une moins bonne généralisation.

- **Conclusion (Fashion-MNIST)**

Mish se distingue comme la meilleure fonction d'activation sur la base de données Fashion-MNIST, tant en apprentissage qu'en validation, sur toutes les métriques observées.

Les fonctions comme Gelu et Swish offrent également de très bonnes performances.

Tableau 4.4 : Les FA les plus performantes pour les trois bases de données.

Base de données	FA les plus performantes (OA, AA, Kappa)	FA les moins performantes	Remarques principales
MNIST	(Swish, LeakyReLU) 99.21%, 99.21%, 99.12	Softsign (98.49%)	Toutes les FA sont proches en performance (<1% d'écart)
SVHN	Gelu : 92.01% , 91.66% , 90.9	Tanh, ELU, Softsign	Gelu se distingue clairement, Softsign est instable
Fashion MNIST	Mish : 99.12% / 99.12% / 99.02	Tanh, Softsign	Mish offre la meilleure convergence et généralisation

4.5 Conclusion

L'analyse menée dans ce chapitre montre clairement que le choix de la fonction d'activation peut influencer les performances d'un réseau de neurones, bien que les écarts restent relativement minimes sur certaines bases comme MNIST où toutes les fonctions testées restent efficaces. Cependant, sur des ensembles de données plus complexes comme SVHN et Fashion-MNIST, les fonctions d'activation modernes en particulier GELU, Mish et Swish se démarquent par une meilleure précision, tant en phase d'apprentissage qu'en validation. Tandis que des fonctions plus classiques, telles que Relu et Tanh, offrent des performances acceptables. En résumé, le chapitre illustre l'importance d'une analyse comparative rigoureuse pour choisir avec pertinence les outils et techniques adaptés aux spécificités de chaque problème, tout en démontrant l'efficacité combinée de MATLAB comme environnement de simulation et de l'optimiseur ADAM pour le réglage des paramètres du modèle.

Conclusion general

Nous avons exploré l'impact de différentes fonctions d'activation sur les performances des réseaux de neurones convolutionnels (CNN) appliqués à la classification d'images. Ce domaine, qui combine intelligence artificielle et traitement d'images.

Notre étude s'est concentrée sur l'analyse comparative de plusieurs fonctions d'activation classiques et modernes, telles que ReLU, leakyrelu, swish, tanh, elu, mish, gelu, softplus, softsign appliqué à un modèle CNN dans notre cas ResNet18. Ces fonctions jouent un rôle crucial dans le comportement d'apprentissage du réseau, affectant directement la convergence, la précision et la capacité à éviter des problèmes tels que la disparition du gradient et la généralisation.

Grâce à MATLAB et à des tests sur des bases de données de référence (MNIST, Fashion-MNIST et SVHN), nous avons tiré plusieurs enseignements clés des résultats expérimentaux :

Les fonctions d'activation, telles que Swish, LeakyReLU, Mish, ELU, Softplus, Gelu se sont révélées particulièrement efficaces en terme de précision de classification en fonction de la base de données utlilisée.

À l'inverse, les fonctions Tanh, ELU, Softsign, malgré leur importance historique, présentent des limites dans les architectures modernes en raison de leur tendance à provoquer une atténuation du gradient.

Le choix optimal de la fonction d'activation dépend également du type de modèle CNN et de la complexité des données traitées.

Ce mémoire met en lumière l’importance stratégique du choix de la fonction d’activation dans la conception de réseaux de neurones efficaces. Nos travaux offrent ainsi des recommandations pratiques à destination des chercheurs et praticiens souhaitant optimiser les performances de leurs modèles CNN pour des tâches spécifiques de classification d’images.

En perspective, l’exploration de fonctions d’activation adaptatives ou issues de techniques d’apprentissage automatique (comme les fonctions apprises ou optimisées automatiquement) pourrait offrir de nouvelles voies d’amélioration. De plus, l’intégration de ces fonctions dans des architectures plus complexes constitue un axe prometteur pour les travaux futurs.

Liste Des Références

[1] Introduction - Artificial Intelligence (AI) - LibGuides at University of Texas at Dallas
[2] https://amt-lab.org/blog/2017/3/a-brief-history-of-artificial-intelligence
[3] Mémoire de Fin d'Etudes La Classification d'images d'insectes ravageurs en utilisant le Deep Learning Réalisé par : LOUNIS Katia MOUSSI Dahbia UNIVERSITE MOULOUD MAMMERI DE TIZI-OUZOU FACULTE DE GENIE ELECTRIQUE ET INFORMATIQUE DEPARTEMENT D'INFORMATIQUE.
[4] Introduction à l'Intelligence Artificielle GI 2020 : M. Benbrahim.
[5] Apprentissage profond | Psychomédia (psychomedia.qc.ca).

[6]] J. Schmidhuber (2015), « Deep learning in neural networks: An overview » [archive], Neural Networks, 61, 85-117.

[7] D. H. Ackley, G. E. Hinton et T. J. Sejnowski (1985), « A learning algorithm for Boltzmann machines », Cognitive Science, 9, 147{169. 590.

[8] « OpenAI lance une version allégée de son générateur automatisé de textes GPT-2 [archive] », sur Le Monde Informatique, 18 février 2019 (consulté le 9 octobre 2024).
[9] Chapitre 7 : Apprentissage profound Pr. Mustapha BOURAHLA, Département d'Informatique, Université de M'Sila.
[10] Artificial Intelligence > Neural Nets (Stanford Encyclopedia of Philosophy)
[11] Les Réseaux de Neurones Artificiels Djillali Liabes University Sidi Bel Abbès https://www.researchgate.net/publication/319939107
[12] Perceptron-Wikipedia http://fr Wikipedia.org.
[13] Pierre Buyssens, Fusion de différents modes de capture pour la reconnaissance du visage appliquée aux e_ transactions DOCTORAT de l'UNIVERSIT´E de CAEN Le 4 janvier 2011.
[14] Optimization technique popularly used in deep learning. http:// medium.com.
[15] [Submitted on 21 Apr 2020 (this version), latest version 4 May 2020 (v2)] AdaX: Adaptive Gradient Descent with Exponential Long Term Memory Wenjie Li, Zhaoyang Zhang, Xinjiang Wang, Ping Luo
[16] RMSProp - Cornell University Computational Optimization Open Textbook - Optimization Wiki
[17] ADAM: A METHOD FOR STOCHASTIC OPTIMIZATION Diederik P. Kingma* University of Amsterdam, OpenAI dpkingma@openai.com Jimmy Lei Ba∗ University of Toronto jimmy@psi.utoronto.ca Adam: A Method for Stochastic Optimization

[18] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," in Advances in neural information processing systems, 2012, pp. 1097–1105.

[19] Szegedy, Christian, et al. "Going deeper with convolutions." Proceedings of the IEEE conference on computer vision and pattern recognition. 2015.

[20] Mémoire de fin d'études Thème Classification des images avec les réseaux de neurones convolutionnels Université Abou Bakr Belkaid Tlemcen Faculté Des Sciences Département D'informatique

[21] L. Stankovi´c, Digital Signal Processing with Selected Topics. Create Space Independent Publishing Platform, An Amazon.com Company, 2015.

[22] Y. LeCun et al, «Gradient-based learning applied to document recognition, » Proceedings of the IEEE, vol. 86, pp. 2278-2324, 1998.

[23] KRIZHEVSKY, Alex, SUTSKEVER, Ilya, HINTON et E. Geoffrey, «ImageNet classification with deep convolutional neural networks, » in Advances in neural information processing systems, pp. 1097-1105, 2012.

[24] Chen, J., Wan, Z., Zhang, J., Li, W., Chen, Y., Li, Y., & Duan, Y. (2021). Medical image segmentation and reconstruction of prostate tumor based on 3D AlexNet. Computer methods and programs in biomedicine, 200, 105878.

[25] C. Szegedy, W. Liu, Y. Jia, P. Sermane, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke et A. Rabinovich, «Going deeper with convolutions, » in Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 1-9, 2015.

[26] A Deep Learning Approach for Automated Diagnosis and Multi-Class Classification of Alzheimer's Disease Stages Using Resting-State fMRI and Residual Neural Networks Farheen Ramzan , Muhammad Usman Ghani Khan , Asim Rehmat , Sajid Iqbal [2], Tanzila Saba , Amjad Rehman , Zahid Mehmood DOI: 10.1007/s10916-019-1475-2

[27] Ian Goodfellow, Yoshua Bengio et Aaron Courville, Deep Learning, MIT Press, 2016 (ISBN 0262035618, lire en ligne [archive]) [détail des éditions], chapitre 7.

[28] BORGI A., AKDAG H. Supervised Learning and Approximate. 2001

[29] Classification supervisée : des algorithmes et leur calibration automatique .Sylvain Arlot. Cours de 3eme année École Centrale de Paris mars 2009.

[30] Akbani, R.Kwek,S,andJapkowicz.N (2004). Applying support vector Machine to imbalnced Datasets. Page35-50

[31] Memoire Approche Exploratoire Sur La Classification Appliquée Aux Images. LamriLaouamer Université Du Québec, Avril 2006

[32] Parrochia. Classifications, histoire et problèmes formels. Cinquièmes Rencontres de la société Francophone de classification SFC'97. Lyon : s.n., Septembre 1997
[33] S. Mannor et al., "K-Means Clustering," in Encyclopedia of Machine Learning, Boston, MA: Springer US, (2011), pp. 563–564
[34] Samuel AMBAPOUR Introduction à l'analyse des données des chiffres, etc.
[35] N. R. Pal, K. Pal, J. M. Keller and J. C. Bezdek. "A possibilistic fuzzy c-means clustering algorithm". IEEE Transactions on Fuzzy Systems 13(4), pp. 517–530, 2005

[36] https://www.analyticsvidhya.com/blog/2020/10/how-does-the-gradient-descentalgorithm-work-in-machine-learning/?utm_source=blog&utm_medium=variants-of-gradientdescent-algorithm

[37] Available: https://atcold.github.io/NYU-DLSP20/fr/week11/11-1/
[38] J. Turian, J. Bergstra, and Y. Bengio, "Quadratic features and deep architectures for chunking," in Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics, vol. Companion Volume: 2009, pp.245–248. [Online]. Available : https://dl.acm.org/citation.cfm
[39] Available: https://www.gabormelli.com/RKB/Softsign_Activation_Function
[40] J. Han and C. Moraga, "The influence of the sigmoid function parameters on the speed of backpropagation learning,," in Natural to Artificial Neural Computation. IWANN 1995. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 1995, pp. 195–201.
[41] R. M. Neal, "Connectionist learning of belief networks," Artificial Intelligence, vol. 56, no. 1, pp. 71–113, 1992. [Online]. Available: https://doi.org/10.1016/0004-3702(92)90065-6
[42] P. Ramachandran, B. Zoph, and Q. V. Le, "Searching for Activation Functions," ArXiv, 2017. [Online]. Available: 1710.05941; http://arxiv.org/abs/1710.05941
[43] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," Nature, vol. 521, no. 7553, pp. 436–444, 2015. [Online]. Available: https://doi.org/10.1038/nature14539
[44] B. Karlik and A. Vehbi, "Performance Analysis of Various Activation Functions in Generalized MLP Architectures of Neural Networks,"International Journal of Artificial Intelligence and Expert Systems (IJAE), vol. 1, no. 4, pp. 111–122, 2011. [Online]. Available: http://www.cscjournals.org/library/manuscriptinfo.php
[45] 7 fonctions d'activation populaires que vous devez connaître dans Deep Learning et comment les utiliser avec Keras et TensorFlow 2 , Les fonctions d'activation des réseaux de neurones Elaborer par : Le Doctorant Adel BRAHAM Lesfonctionsdactivationdesrseauxdeneurones.pdf
[46] Goodfellow, Y. Bengio, and A. Courville, "Deep learning." MIT Press, 2016. [Online]. Available: http://www.deeplearningbook.org

[47] Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," 2012. [Online]. Available : http://citeseerx.ist.psu.edu/viewdoc/summary

[48] M. D. Zeiler, M. Ranzato, R. Monga, M. Mao, K. Yang, Q. V. Le, and G. E. Hinton, "On rectified linear units for speech processing," in International Conference on Acoustics, Speech and Signal Processing. IEEE, 2013, pp. 3517–3521, IEEE. https://doi.org/10.1109/ICASSP.2013.6638312.
[49] Nair and G. E. Hinton, "Rectified linear units improve restricted boltzmann machines," Haifa, 2010, pp. 807–814. [Online]. Available: https://dl.acm.org/citation.cfm
[50] G. E. Dahl, T. N. Sainath, and G. E. Hinton, "Improving deep neural networks for LVCSR using rectified linear units and dropout," in International Conference on Acoustics, Speech and Signal Processing. IEEE, 2013
[51] X. Glorot, A. Bordes, and Y. Bengio, "Deep Sparse Rectifier Neural Networks," in International Conference on Machine Learning, 2011. [Online].Available: http://proceedings.mlr.press/v15/glorot11a/glorot11a.pdf
[52] A. Maas, A. Hannun, and A. Ng, "Rectifier Nonlinearities Improve Neural Network Acoustic Models," in International Conference on Machine Learning(icml), 2013.
[53] K. He, X. Zhang, S. Ren, and J. Sun, "Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification," arXiv, 2015.[Online]. Available: http://arxiv.org/abs/1502.01852
[54] Available : https://arxiv.org/abs/1606.08415
[55] B. Xu, N. Wang, H. Kong, T. Chen, and M. Li, "Empirical Evaluation of Rectified Activations in Convolution Network," arXiv, 2015. [Online].Available: https://arxiv.org/abs/1505.00853

[56] X. Jin, C. Xu, J. Feng, Y. Wei, J. Xiong, and S. Yan, "Deep Learning with S-shaped Rectified Linear Activation Units,," arXiv, pp. 1737–1743, 2015.[Online]. Available: https://arxiv.org/pdf/1512.07030.pdf
[57] D. A. Clevert, T. Unterthiner, and S. Hochreiter, "Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs)," arXiv, 2015.[Online]. Available: http://arxiv.org/abs/1511.07289
[58] L. Trottier, P. Giguere, and B. Chaib-draa, "Parametric Exponential Linear Unit for Deep Convolutional Neural Networks," arXiv, 2018. [Online]. Available: https://arxiv.org/pdf/1605.09332v1.pdf
[59] A Self Regularized Non-Monotonic Neural Activation Function. Available : https://arxiv.org/vc/arxiv/papers/1908/1908.08681v1.pdf

[60] T Zgheib, H Borges, V Feldman, T Guntz, C Di Loreto, O Desmaison, F Corduant: Détection d'objets en temps réel: Entraînement de réseaux de neurones convolutifs sur images réelles et synthétiques