جامعة سيدي محمد بن عبد الله بفاس
UNIVERSITÉ SIDI MOHAMED BEN ABDELLAH DE FES

كلية العلوم والتقنيات فاس
FACULTÉ DES SCIENCES ET TECHNIQUES DE FÈS

Projet de Fin d'Etudes

Licence Sciences et Techniques Génie Informatique

Application pour La Reconnaissances des caractères manuscrits

FACULTÉ DES SCIENCES DHAR EL MAHRAZ
UNIVERSITÉ SIDI MOHAMED BEN ABDELLAH

Lieu de Stage : *Faculté des sciences DHAR EL MAHRAZ*

Réalisé par :

Abderrahman El Alami

Hamza EL Alami

Encadré par :

Mr. Khalid Satori

Mr. Ismail Elbatteoui

Mr. Jamal Kharroubi

Soutenu le 10/06/2021 devant le jury composé de :

Pr. J. Kharoubi

Pr. L. Lamrini

Pr. F.Mrabti

Année Universitaire 2020-2021

Remerciement

On tient à exprimer nos remerciements à tous ceux qui ont rendu ce travail possible. Leurs aides précieuses, leurs conseils fructueux et leurs encouragements, tout au long de l'élaboration de ce projet de fin d'études, nous a permis de le réaliser dans la meilleure considération.

On rendre un hommage particulier à nos encadrants:

Mr Ismail EL batteoui & Mr Khalid Satori & Mr Jamal Kharoubi

Pour leurs soutiens et leurs conseils précieux.

Aux membres du jury qui ont bien voulu nous honorer de leur présence d'évaluer notre travail. Un grand merci à toutes les personnes qui nous ont soutenus de près ou de loin au cours de la réalisation de ce modeste travail.

Résumé

Dans notre travail on a utilisé Les réseaux de neurones pour la classification des images en général et la reconnaissance des caractères en particulier. Les résultats obtenus ont montré que le choix du nombre des neurones et la profondeur du réseau ont une grande influence pour avoir des meilleurs résultats.

ABSTRACT

In our work we have used Multilayer neural networks for the classification of images in general and the recognition of characters in particular. The results obtained showed that the choice of the number of neurons and the depth of the network have a great influence on obtaining better results

Tables des Figures :

Introduction Générale

Les réseaux de neurones sont des modèles théoriques de traitement de l'information inspirés des observations relatives au fonctionnement des neurones biologiques et du cortex cérébral. Le domaine des réseaux de neurones n'est pas nouveau car il a son origine dans des travaux conduits durant les années 40 (modèle de Hebb [**1]** pour l'évolution des connexions synaptiques).

Ces travaux conduisirent au modèle du perceptron dans les années 60 (modèle qui a principalement été appliqué à la reconnaissance de caractères **[2]**). Mais ce n'est qu'à partir de 1986 que la recherche dans ce domaine a connu une expansion importante du fait de la publication de modèles de réseaux et d'algorithmes d'apprentissage suffisamment efficaces pour résoudre des problèmes réalistes et complexes.

Dans la fin des années 80 Yan le Cun **[3]** a développé un type de réseau particulier qui s'appelle le réseau de neurones convolutionnel, ces réseaux sont une forme particulière de réseau neuronal multicouche (chapitre3) dont l'architecture des connexions est inspirée de celle du cortex visuel des mammifères. Par exemple, chaque élément n'est connecté qu'à un petit nombre d'éléments voisins dans la couche précédente. En 1995, Yan le Cun et deux autres ingénieurs ont développé un système automatique de lecture de chèques qui a été déployé largement dans le monde. À la fin des années 90, ce système lisait entre 10 et 20 % de tous les chèques émis aux États-Unis. Mais ces méthodes étaient plutôt difficiles à mettre en œuvre avec les ordinateurs de l'époque, et malgré ce succès, les réseaux convolutionnels et les réseaux neuronaux plus généralement ont été délaissés par la communauté de la recherche entre 1997 et 2012. Aujourd'hui, cette technique est partout. Depuis les assistants virtuels, tels que Home de Google ou Alexa d'Amazon, qui commencent à remplacer efficacement le travail d'assistants et de secrétaires humains. En Chine, les réseaux profonds sont utilisés quotidiennement pour reconnaître les visages de millions de citoyens qui passent devant les caméras de surveillance.

La classification statistique réside dans l'identification des catégories à laquelle un nouveau élément appartient, sur la base d'un data set d'entrainement de données contenant des observations dont la catégorie est connue .par exemple attribuer un courrier électronique donné au classe « spam » ou « no spam ».

La classification des images consiste à attribuer une classe à une image à l'aide d'un système de classification en se basant sur ce qu'on appelle l'apprentissage. On retrouve ainsi la classification d'objets, de scènes, de textures, la reconnaissance de visages, d'empreintes digitale et de caractères. Il existe deux principaux types d'apprentissage l'apprentissage supervisé et l'apprentissage non-supervisé. Dans l'approche supervisée, chaque image est associée à une étiquette qui décrit sa classe d'appartenance. Dans l'approche non supervisée les données disponibles ne possèdent pas d'étiquettes.

Pour classifier notre image nous avons utilisé la méthode de réseaux de neurone. Ils sont issus d'une tentative de conception d'un modèle mathématique très simplifié du cerveau humain en se basant sur notre façon d'apprendre et de corriger nos erreurs. Il existe plusieurs types de réseaux de neurones tels que les "perceptrons", les réseaux à fonctions de base radiales et les réseaux récurrents. Parmi eux, les perceptrons à (feed-forward) et entraînés par rétropropagation (backpropagation) ont eu un succès important dans

plusieurs applications**.** Nous avons choisi le perceptron multicouche pour la reconnaissance de caractères manuscrits.

Pour évaluer les performances de notre application, nous avons utilisé deux bases de données (La base de données MNIST et la basse de données EMNIST).

Notre rapport est organisé de la manière suivant :

- Dans le chapitre 2 nous présentons, les caractéristiques d'images et les notions de base de la classification des caractères manuscrites.
- Le troisième chapitre est consacré à la description des réseaux de neurones artificiels.
- La partie d'apprentissage ainsi que La méthode de la Rétropropagation du Gradient sont représenté au Chapitre 3.
- Dans le quatrième chapitre, on va monter la partie expérimentale de notre travail

Chapitre 1 : Reconnaissances Des caractères manuscrits

I. Notion De Base

1. Définition d'Image

Une image est une représentation planaire d'une scène ou d'un objet situé en général dans un espace tridimensionnel, elle est issue du contact des rayons lumineux provenant des objets formants la scène avec un capteur (caméra, scanner, rayons X, ...). Il ne s'agit en réalité que d'une représentation spatiale de la lumière. L'image est considérée comme un ensemble de points auquel est affectée une grandeur physique (luminance, couleur). Ces grandeurs peuvent être continues (image analogique) ou bien discrètes (images digitales). Mathématiquement, l'image représente une fonction continue IF, appelée fonction image, de deux variables spatiales représentée par IF(x, y) mesurant la nuance du niveau de gris de l'image aux coordonnées (x, y). **[4]**

2. Caractéristique d'Image

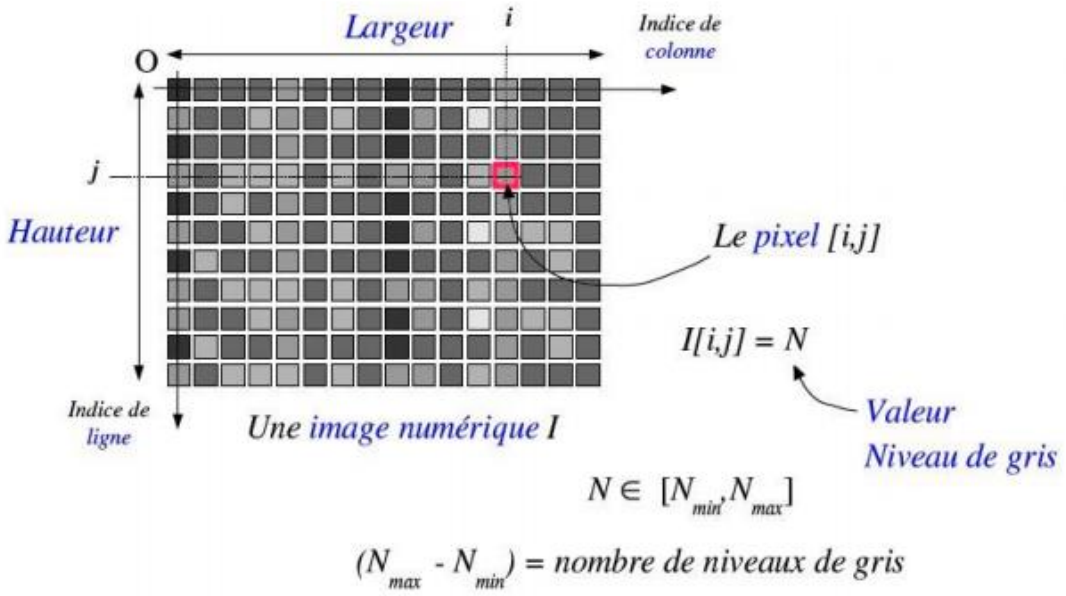

Figure 1 : Représentation des caractéristiques d'Image.

3. Pixel

On appelle dimension, le nombre de points (pixel) constituant l'image, c'est à dire sa (dimension informatique). Cette dernière se présente sous forme de matrice dont les éléments sont des valeurs numériques représentatives des intensités lumineuses (pixels). Le nombre de lignes de cette matrice multiplié par le nombre de colonnes nous donne le nombre total de pixels dans une image **[5].**

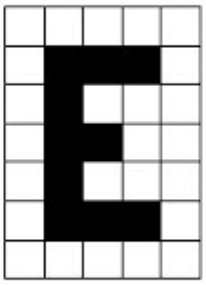

Figure 2: représentation du caractère E sous forme de Pixel.

4. Dimension & Résolution

La dimension est la taille de l'image. Elle se présente sous forme d'une matrice dont les éléments sont des valeurs numériques représentatives des intensités lumineuses (pixels). Le nombre de lignes de cette matrice multipliée par le nombre de colonnes nous donne le nombre total de pixels dans une image. Par contre, la résolution est la clarté ou la finesse de détails atteinte par un moniteur ou une imprimante dans la production d'images .Sur les moniteurs d'ordinateur, la résolution est exprimée en nombre de pixels par unité de mesure (pouce ou centimètre). On utilise aussi le mot résolution pour désigner le nombre total de pixels horizontaux et verticaux sur un moniteur. Plus ce nombre est grand, plus la résolution est meilleure.

5. Niveau de Gris

C'est la valeur d'intensité lumineuse d'un pixel. Cette valeur peut aller du noir (0) jusqu'au blanc (255) en passant par les nuances qui sont contenues dans l'intervalle [0, 255]. Elle correspond en fait à la quantité de la lumière réfléchie. Pour 8 bits, on dispose de 256 niveaux de gris dont 40 sont reconnus à l'œil nue. Plus le nombre de bit est grand plus les niveaux sont nombreux et plus la représentation fiable **[4].**

II. Reconnaissances des Caractère Manuscrites

Les méthodes les plus performantes pour reconnaitre un caractère manuscrit sont basées sur des méthodes d'apprentissage statistique : leur principe commun est de fonder leur prédiction sur la comparaison de l'image du caractère manuscrit à classer à d'autres images de caractères manuscrits pour lesquels la nature du caractère est connue. Typiquement, si une image arrive et qu'elle est très similaire à l'image de caractère 'A' de notre base d'apprentissage, l'algorithme classera l'image dans la catégorie 'A'.

1. Base de données NIST

La base de données du NIST[1] est composée d'images extraites de formulaires de recensement. Elle contient à la fois des caractères isolés et du texte écrits par 3.600 scripteurs pour un total de 800.000 images de résolution 300dpi. La dernière

[1] NIST : National Institue of Standards and Technology

version, décrite sur le site **[6],** est la NIST Spécial Database 19 qui fait suite aux versions 3 et 7.

2. Base de données EMNIST

EMNIST[2] Est un ensemble de caractères manuscrits dérivés de la base de données spéciale NIST 19 et convertis en un format d'image de 28x28 pixels et une structure d'ensemble de données qui correspond directement à l'ensemble de données MNIST. Elle est constituée de 800 000 caractères manuscrits vérifiés et étiquetés de près de 3700 écrivains qui a rempli un formulaire.

Figure 3: un exemple de chiffres de la base EMNIST.

À l'origine, NIST SD 19 avait deux schémas d'étiquetage différents :

By_Classe : dans ce schéma, les classes sont les chiffres [0-9], les lettres minuscules [a-z] et les lettres majuscules
[A–Z]. Ainsi, il existe 62 classes différentes.

By_Merge : ce schéma répond au fait que certaines lettres sont assez similaires dans leur minuscule et les variantes en majuscules, ainsi les deux classes peuvent être fusionnées. En particulier, ces lettres sont 'c', 'i', 'j', 'k', 'l', 'm', 'o', 'p', 's', 'u', 'v', 'w', 'x', 'y' et 'z'. Ce schéma contient 47 classes.
Ces schémas ont été portés sur EMNIST.

De plus, en plus de ces deux ensembles de données, EMNIST a ensembles de données supplémentaires générés :

Equilibré (Balanced) : les ensembles de données By_Class et By_Merge sont très déséquilibrés en ce qui concerne les lettres, un fait qui pourraient avoir un impact négatif sur les performances de classification. Cet ensemble de données prend le By_Merge et réduit le nombre d'instances de 814.255 (nombre total d'échantillons dans le NIST 19) à seulement 131 600, garantissant qu'il y a un nombre égal d'échantillons par chaque étiquette.

[2] EMNIST : an Extension of MNIST to handwritten letters

Chiffres (Digits) : similaires à MNIST, mais d'une autre source et avec 280 000 instances.

Lettres (Letters): cet ensemble de données contient un mélange de lettres minuscules et majuscules, contenant ainsi 26 classes et un total de 145 600 échantillons.

3. Base de données MNIST

La base de données MNIST[3] est une base de données de chiffres écrits à la main, issues d'une base de données antérieure, appelée simplement NIST. Elle est constituée de 70 000 chiffres manuscrits au format 28 pixels par 28 pixels (60000 images d'apprentissage et 10000 images de test) où chaque pixel est représenté par un niveau de gris allant de 1 à 256.

Cette base est gratuite et disponible sur le site **[7] .**Ce site donne aussi les résultats d'évaluation de plusieurs techniques classiques de reconnaissance de formes sur cette base. Quelques échantillons sont présentés **Figure 4** Nous utiliserons cette base pour entrainer et tester notre réseau pour la reconnaissance des chiffres manuscrits.

Figure 4:un exemple de chiffres de la base MNIST.

III. Méthodes de Reconnaissances de caractères

Une Apprentissage machine « machine Learning en Anglais » est une forme de L'intelligence Artificiels qui permet à un Système d'apprendre à partir de données et non à l'aide d'une programmation explicite.

L'apprentissage automatique est utilisé dans un large spectre d'applications pour doter des ordinateurs ou des machines de capacité d'analyser des données d'entrée comme la perception de leur environnement (vision, Reconnaissance de formes tels des visages, langages naturels, caractères dactylographiés ou manuscrits …..).

Cependant, l'apprentissage automatique n'est pas un processus simple. Au fur et à mesure que les algorithmes ingèrent les données de formation, il devient possible de créer des modèles plus précis basés sur ces données. Un modèle de machine Learning est le résultat

[3] MNIST : Mixed National Institue of Standards and Technology

généré lorsque vous entraînez votre algorithme d'apprentissage automatique avec des données.

Il existe plusieurs algorithmes d'apprentissages :

- La régression logistique.
- Les machines à vecteur de support.
- La méthode de K plus proches voisins.
- Les algorithmes génétiques et la programmation génétique.
- **Les réseaux de neurones.**

Dans notre Projet on a utilisé les Réseaux de neurones comme un algorithme d'apprentissage.

Chapitre 2 : Les Réseaux De Neurones Artificiels

I. Introduction

Les réseaux de neurones ont été appliqués avec succès à l'apprentissage de tâches de classification et d'approximation de fonctions Ils peuvent modéliser et approximer des systèmes complexes et difficiles, en utilisant une modélisation conventionnelle mathématique.

Dans ce chapitre, une description générale des réseaux neuronaux artificiels et des aspects liés à leur fonctionnement sera entamée. Nous présenterons des généralités sur le réseau neuronal, telles que la définition de ses structures, les fonctions d'activation et les différentes architectures de ces réseaux.

II. Réseaux De Neurones Artificiels

1. Définition

Un Réseaux de Neurones artificiels, ou bien « Artifical neural Network » est un système informatique, dont le fonctionnement est claque sur celui des neurones biologiques, c'est à dire, le fonctionnement du cerveau et du System nerveux. Les Réseaux de neurones artificiels permettent aux machines de réfléchir et penser comme les humains et de résoudre des problèmes complexes d'une façon autonome.

Avec une telle puissance, il est possible d'entraîner la machine sur des tâches bien plus avancées :

- La reconnaissance d'objets et reconnaissance faciale
- L'analyse de sentiments
- L'analyse du langage naturel
- La création artistique
- Etc…

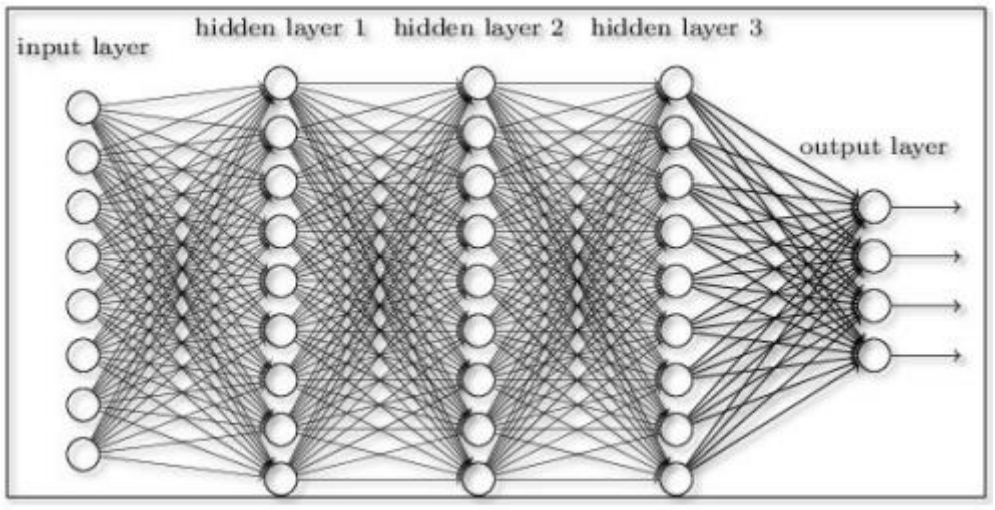

Figure 5:Structure des réseaux De Neurone artificiels.

2. Neurone Formels

Un neurone formel est un modèle mathématique. Il est utilisé dans le cadre des recherches sur l'intelligence artificielle, principalement en association avec d'autres neurones formels pour former un réseau de neurones. **[8]**

Il contient plusieurs formules mathématiques afin de reproduire le plus fidèlement possible le fonctionnement d'un neurone avec ses différentes entrées (dendrites), leurs importances (coefficient de pondération) et sa sortie (axone) et ainsi comprendre son potentiel d'interaction avec les autres neurones. Il est caractérise par :

- Des entrées ei et leur poids synaptique wi
- Une fonction d'activation (de transfert) f
- Une sortie y

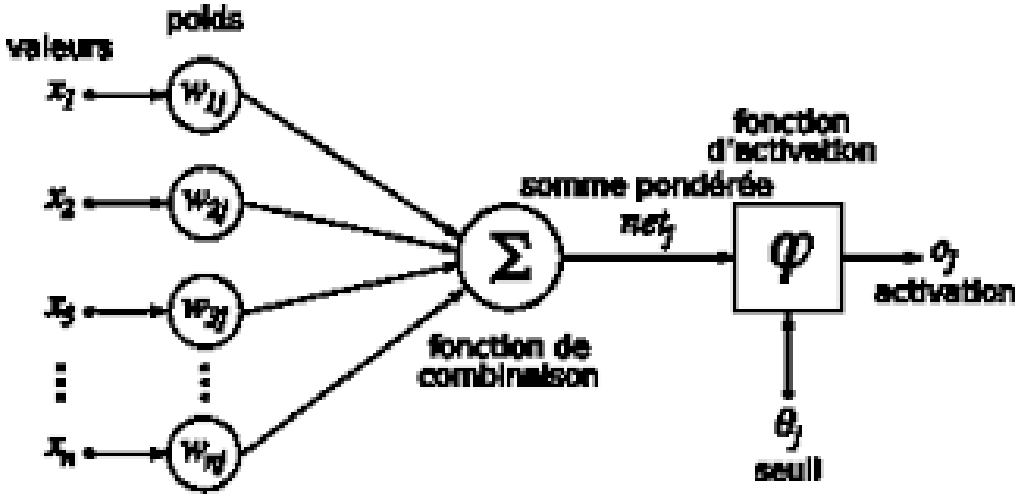

Figure 6:Neurone Formels.

3. Perceptron

Le réseau de neurones le plus simple qui existe porte le nom de perceptron (Réseau de neurones à 1 neurone).

Les entrées de neurones sont les x multipliée par de paramètres (les poids » à apprendre. Le calcul effectué par le neurone peut être divisé en deux étapes :

1. Le neurone calcule la somme Z de toutes les Entrées $\boldsymbol{Z = \sum(xi * wij)}$
2. Le neurone posse z dans sa fonction d'activation

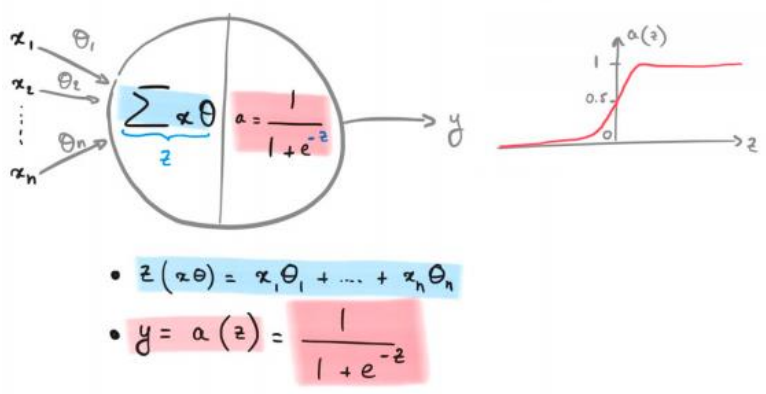

Figure 7 : Perceptron.

Note :

On utilise souvent d'autres fonctions d'activation que la fonction sigmoïde pour simplifier le calcul du gradient et ainsi obtenir des cycles d'apprentissage plus rapides :

- La fonction tangente hyperbolique tanh (z)
- La fonction $Relu$(z)

4. Fonctions d'Activation

La fonction d'activation (fonction de transfert) joue un rôle très important dans le comportement des neurones, elle contient un paramètre sur la somme des entrées pondérées en plus du seuil d'activation. Il en existe plusieurs types dont la nature est déterminée en fonction du réseau, **le tableau 1** résume ces différents types, les trois les plus utilisées sont les fonctions seuil, linéaire et sigmoïde **[9].**

Tableau 1 : Les différents Fonctions d'activation.

Seuil	**a=0** si $n < 0$ **a=1** si $n > 0$
Seuil symétrique	**a=-1** si $n < 0$ **a=1** si $n \geq 0$
Linéaire	**a=n**
Linéaire saturée	**a=0** si $n < 0$ **a=n** si $0 \leq n \leq 1$ **a=1** si $n > 0$
Linéaire saturée symétrique	**a=-1** si $n < -1$ **a=n** si $-1 \leq n \leq 1$ **a=1** si $n > 1$
Linéaire positive	**a=0** si $n < 0$ **a=n** si $n \geq 0$
Sigmoïde	$a = \frac{1}{1 + e^{-n}}$
Tangente hyperbolique	$a = \frac{e^{n} - e^{-n}}{e^{n} + e^{-n}}$

5. Perceptron multicouche

Le perceptron multicouche (Multi Layer Perceptron MLP) est un réseau comportant plusieurs couches, chaque neurone d'une couche étant totalement connecté aux neurones de la couche suivante. Chaque neurone est un automate linéaire généralisé dont la fonction de transfert est supposée sigmoïdale **(Figure 8)**. Une information circule de la couche d'entrée vers la couche de sortie uniquement ; il s'agit donc d'un réseau de type feedforward. Chaque

couche est constituée d'un nombre variable de neurones, les neurones de la couche de sortie correspondant toujours aux sorties du système **[10].**

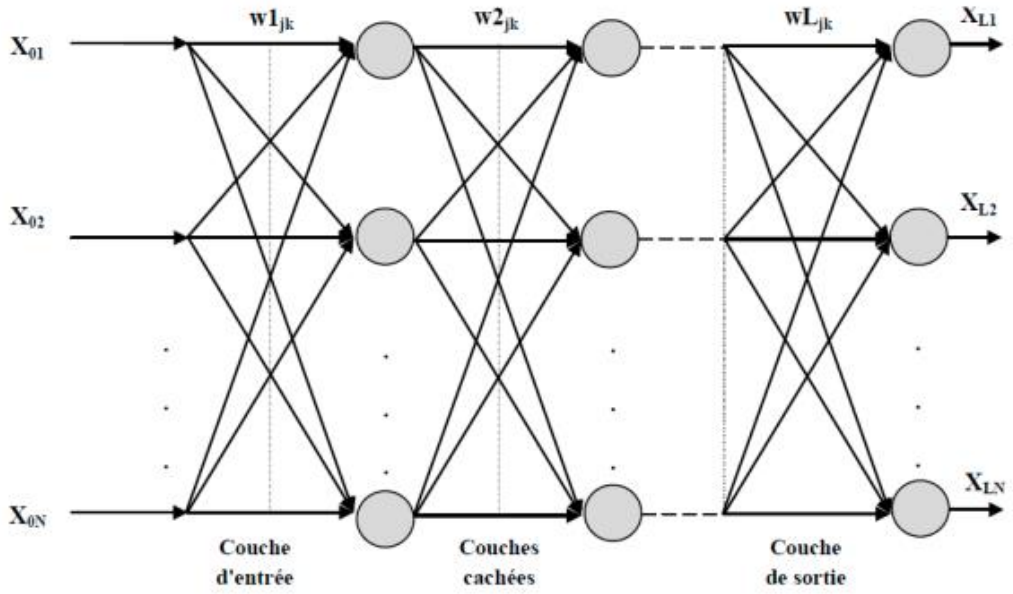

Figure 8:Perceptron multicouche.

Pour créer un Réseaux de Neurones, il suffit de développer plusieurs de ces perceptrons et de les connecter les uns aux autres d'une façon particulière :

• On réunit les neurones en colonne (on dit qu'on les réunit en couche, en layer). Au sein de leur colonne, les neurones ne sont pas connectés entre eux.

• On connecte toutes les sorties des neurones d'une colonne à gauche aux entrées de tous les neurones de la colonne de droite qui suit.

6. Exemples de Réseaux de neurones

a. Exemple 1

On peut ainsi construire un réseau avec autant de couches et de neurones que l'on veut. Plus il y a de couches, plus on dit que le réseau est profond (deep) et plus le modèle devient riche, mais aussi difficile à entraîner.. Voici un exemple d'un réseau à 5 neurones (et 3 couches). Tous les couches entre la couche d'entrée et la couche de sortie sont dits cachés car nous n'avons pas accès à leur entrées/sorties, qui sont utilisées par les couches suivants.

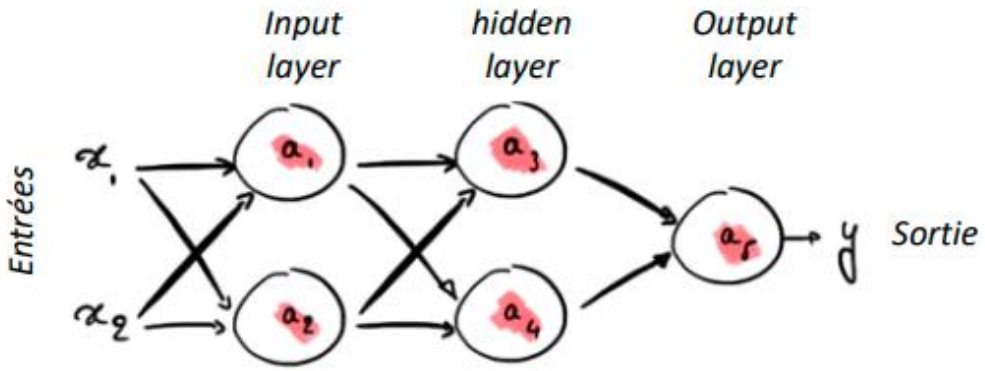

Figure 9:Réseaux de neurones à 5 neurones et 3 couches.

b. Exemple 2

Un réseau plus simple (à 3 neurones) nous donnerait la sortie $\boldsymbol{a3}$ = ($\boldsymbol{\theta 1 a1}$ + $\boldsymbol{\theta 2 a2}$) où $\boldsymbol{\theta 1}$ et $\boldsymbol{\theta 2}$ sont les coefficients liés aux connections entre neurones $\boldsymbol{a1} \rightarrow \boldsymbol{a3}$ et $\boldsymbol{a2} \rightarrow \boldsymbol{a3}$. Ce sont les paramètres de notre modèle. Dans le réseau suivant, on a donc 6 paramètres.

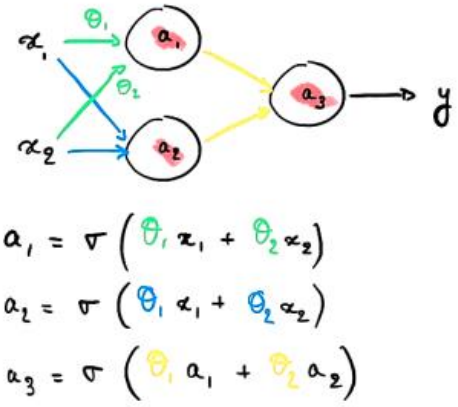

Figure 10:Réseaux de neurones à 3 neurones et 2 couches.

c. Problème XOR

Pour bien comprendre les Réseaux de neurones et de familiariser avec les notions de base du Réseaux, notre encadrant nous a demander de crée un Réseaux de neurones qui permet de résoudre **le problème XOR.**

Le réseau ce qu'on a réalisé contient 5 neurones comme vous voyez dans la figure ci-dessous **(figure 11).**

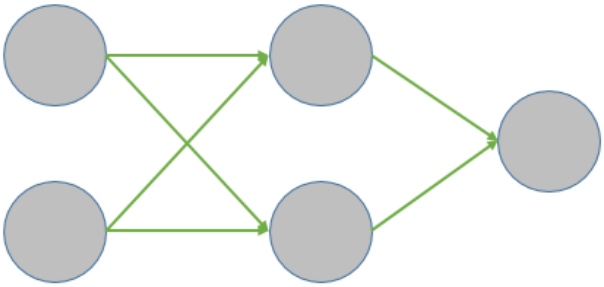

Figure 11: RNA[4] pour résoudre le problème XOR.

III. Type des Réseaux De Neurones Artificiels

1. Réseaux De Neurones uniderctionnel (feed forward)

C'est un cas particulier des réseaux de neurones dont les signaux vont toujours de la gauche vers la droite, ils ne vont donc que dans une seule direction

2. Réseaux de neurones convulsifs

Un réseau a convolution est un Réseau très utilise dans l'analyse et la classification de l'image, car il permet d'extraire des caractéristiques ce qu'est incapable de faire un réseau de neurone classique.

Un réseau de neurones à convolution est essentiellement composé de 4 parties :

- **Convolution**
- **Non-linéarité (ReLu)**
- **Pooling**
- **Classification**

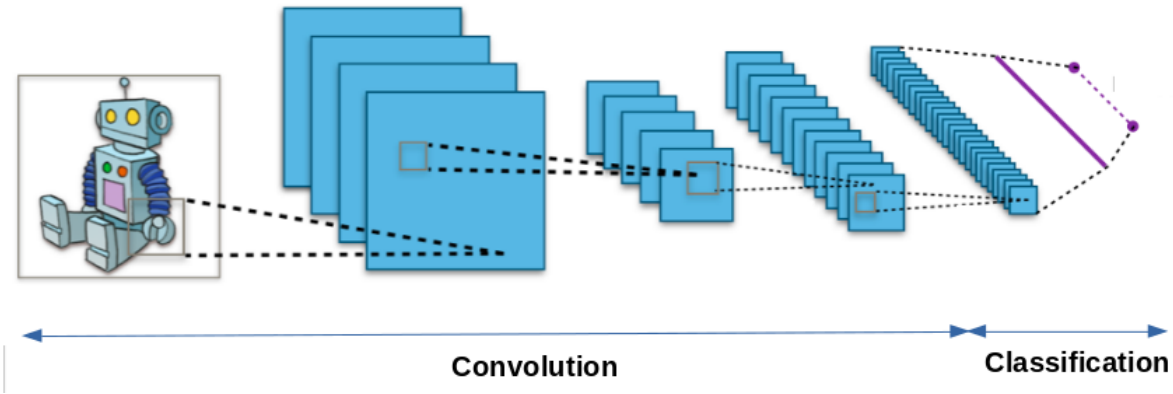

Figure 12:Réseaux de Neurones convulsifs.

[4] **R**éseaux de **N**eurones **A**rtificiels

- **Couche De Convolution (CONV) :**

La couche de convolution est la composante clé des réseaux de neurones convolutionnels, et constitue toujours au moins leur première couche. Son but est de repérer la présence d'un ensemble des caractéristiques dans les images reçues en entrée. il s'agit de faire glisser une matrice pardessus une image, et pour chaque pixel, utiliser la somme de la multiplication de ce pixel par la valeur de la matrice. Cette technique nous permet de trouver des parties de l'image qui pourraient nous être intéressantes.

- **Couche de Correction (RELU) :**

Relu[5] est une fonction qui doit être appliquée à chaque pixel d'une image après convolution, il remplace toutes les valeurs négatives reçues en entrées par des zéros. En ne modifiant pas les données positives, Relu n'impacte pas les caractéristiques misent en évidence par la convolution.

- **Couche de pooling (Pooling Layer) :**

Pooling est le dernier outil utilisé par les réseaux de neurones à convolution, Celui-ci est utilisé afin de réduire la taille d'une couche tout en s'assurant que les éléments importants d'une image sont gardés.

- Le regroupement spatial est également appelé sous-échantillonnage ou sous-échantillonnage qui réduit la dimensionnalité de chaque carte mais conserve des informations importantes. La mise en commun spatiale peut être de différents types :
- - **Regroupement maximale (max pooling) :** La regroupement maximale (max pooling) prend le plus grand élément de la carte des caractéristiques (feature map) rectifiée.
- - **Regroupement moyenne (average pooling) :** La regroupement moyenne (average pooling) prend la moyenne de tous les éléments de la carte des caractéristiques (feature map) rectifiée.

IV. Conclusion

Une fois qu'on a notre Réseau de neurone ce qu'il va falloir faire c'est l'entérinée afin d'ajuster les poids pour que le Résultats du réseau soit cohérant avec ce qu'on lui a donné, **C'est la phase d'apprentissage**.

[5] RELU : Rectified Linear Unit pour une opération non linéaire

Chapitre 3 : Apprentissages des Réseaux de Neurones Artificiels

I. Introduction

L'apprentissage est la propriété la plus intéressante des réseaux neurones. Une caractéristique des réseaux de neurones est leur capacité à apprendre « Par exemple à reconnaître une lettre, un son... ». Mais cette connaissance n'est pas acquise dès le départ.

L'apprentissage, pour les réseaux de neurones formels, consiste à calculer les paramètres de telle manière que les sorties du réseau de neurones soient, pour les exemples utilisés lors de l'apprentissage, aussi proches que possible des sorties « désirées », qui peuvent être le code de la classe à laquelle appartient la forme que l'on veut classer.

II. Les Algorithmes d'Apprentissages

Les Algorithmes d'apprentissage permet se catégoriser selon le mode d'apprentissage qu'ils empilaient, Il existe plusieurs modes d'apprentissage, comme par exemple : l'apprentissage supervisé, l'apprentissage non-supervisé et l'apprentissage par renforcement.

- **Apprentissage supervisé :**

Dans ce cas, la connaissance à priori de la sortie désirée est nécessaire. On présente au réseau le vecteur d'entrée puis sa sortie est calculée et comparée à la sortie désirée, ensuite les poids sont ajustés de façon à réduire l'écart entre elles. Cette procédure est répétée jusqu'à ce qu'un critère de performance soit satisfait. **La figure 12** illustre le processus d'apprentissage supervisé **[11].**

Parmi les algorithmes supervisés, on distingue **les algorithmes de classification** (prédictions non-numériques) et **les algorithmes de régression** (prédictions numérique).

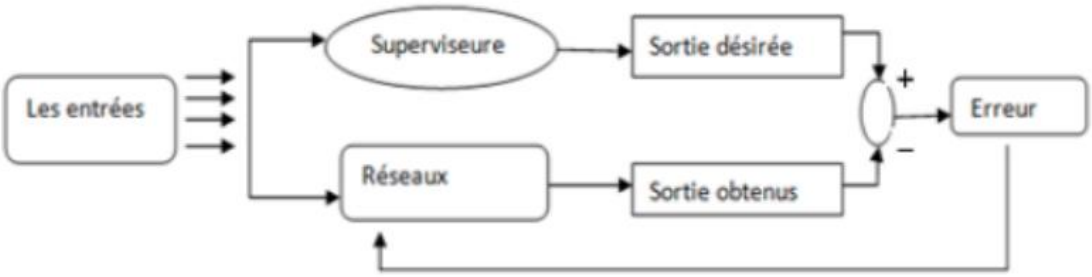

Figure 13:Apprentissage supervisé.

- **Apprentissage non- supervisé :**

L'apprentissage non supervisé consiste à entraîner le modèle sur des données sans étiquettes. La machine parcourt les données sans aucun indice, et tente d'y découvrir des motifs ou des tendances récurrentes.

Parmi les modèles non-supervisés, on distingue **les algorithmes de clustering** (pour trouver des groupes d'objets similaires), **d'association** (pour trouver des liens entre des objets) et **de réduction dimensionnelle** (pour choisir ou extraire des caractéristiques).

Figure 14:Apprentissage non-supervisé.

- **Apprentissage par renforcement :**

Dans ce cas de figure, l'algorithme apprend en essayant encore et encore d'atteindre un objectif précis. Il pourra essayer toutes sortes de techniques pour y parvenir. Le modèle est récompensé s'il s'approche du but, ou pénalisé s'il échoue.

En tentant d'obtenir **le plus de récompenses possible**, il s'améliore progressivement. En guise d'exemple, on peut citer le programme **Alpha GO**[6] qui a triomphé du champion du monde de jeu de Go. Ce programme a été entraîné par renforcement.

III. Les Paramètres d'apprentissage

Tous les modèles de réseaux de neurones requièrent un apprentissage. Plusieurs types d'apprentissages peuvent être adaptés à un même type de réseau de neurones. Les critères de choix sont souvent la rapidité de convergence ou les performances de généralisation. L'efficacité d'apprentissage dépend de plusieurs paramètres :

- **Taux d'apprentissage η:**

Ce paramètre détermine la vitesse de convergence. Si la valeur de démarrage de η est grande, alors on aura un apprentissage très rapide mais au prix de la création d'oscillations dans l'erreur totale moyenne qui empêcheront l'algorithme de converger vers le minimum désiré. Le réseau devient instable. Dans la plupart des cas, si la fonction d'erreur possède plusieurs minimums locaux, le réseau subira un blocage dans l'un d'eux. Toutes ces conditions nous obligent à commencer l'apprentissage avec une petite valeur de η, si on veut atteindre un minimum global, même si l'apprentissage est long.

- **Seuil de tolérance**

Ce paramètre critique détermine la précision dans la réponse du réseau de neurones. La phase d'apprentissage est souvent arrêtée lorsque l'erreur calculée sur l'ensemble de la base d'apprentissage est inférieure à un seuil déterminé par l'utilisateur. Il est possible d'arrêter l'apprentissage en fixant une limite au nombre d'itérations.

IV. La Rétroproagation du Gradient

La Rétroproagation est l'un des méthodes les plus utilise pour entrainer un réseau de neurones artificiels.

[6] Alpha go : programme informatique capable de jouer au jeu de go développer par l'entreprise britannique DeepMind et racheté en 2014 par Google

L'algorithme de descente de gradient ajuste des paramètres pour minimiser une fonction de coût, qui quantifie l'erreur entre la réponse estimée et la bonne réponse sur des données d'entraînement. Ces paramètres sont modifiés itérativement en soustrayant le gradient de la fonction de coût. Cet algorithme nécessite donc de calculer la variation du coût à la sortie du réseau relativement à la variation des paramètres du réseau.

Le gradient est normalement calculé en moyenne sur un batch qui contient toutes les données d'entraînement, ce qui nécessite beaucoup de calculs. L'algorithme de descente de gradient stochastique [1] Réduit ce batch à un seul exemple pris au hasard dans la base de données. Dans le cas où la fonction de coût est strictement convexe, on démontre que l'algorithme de descente de gradient converge vers un minimum unique.

L'algorithme de descente de gradient stochastique a une convergence beaucoup plus lente que l'algorithme par batch, car chaque itération est bruitée par la variabilité du gradient qui dépend de l'exemple particulier qui a été choisi. Cependant, lorsque la base de données est suffisamment grande, cet algorithme peut nécessiter moins d'opérations que la descente de gradient par batch pour approcher le minimum de la fonction de coût.

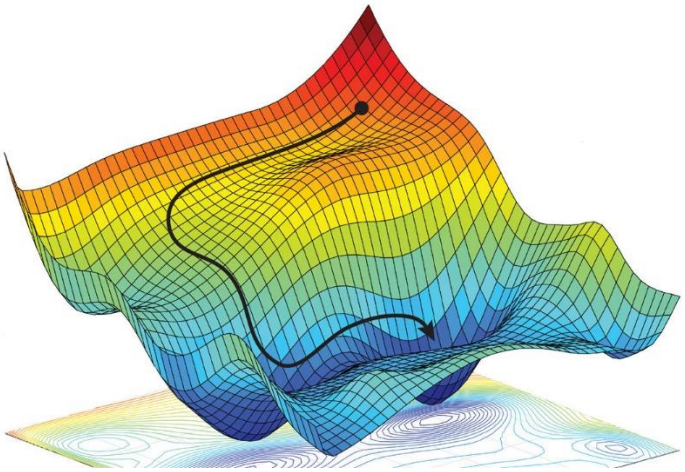

Figure 15:Descente de gradient.

Le gradient de la fonction d'erreur permet de savoir dans quelle direction il faut modifier les paramètres du réseau afin de réduire l'erreur sur l'ensemble des données d'entraînement **[12].**

V. Algorithme de Rétropropagation du Gradient

Le réseau de neurones multicouches contient $N0$ entrées, L-1 couche cachées de Ni neurones et une couche de sortie contenant NL neurones **[13].** L'indice de la couche est i , Xik est la sortie du neurone K de la couche i , $Wijk$ est le poids reliant la sortie $xi{-}1j$ au neurone K de la couche i , Ni est le nombre de neurones dans la couche i **(voir7).** Avec ces notions, la sortie Xik du neurone (i ,k)sera donnée par :

$$\boldsymbol{y_{ik} = \sum_{j=1}^{N(i-1)} W_{iJK} x_{i-1} + b_{ik}} \qquad \textbf{(1)}$$

$$\boldsymbol{X_{ik} = f(y_{ik})} \qquad \textbf{(2)}$$

C'est un algorithme supervisé dont l'objectif est d'ajuster les poids du réseau MLP de façon à minimiser une fonction de coût différentiable, telle que l'erreur quadratique entre la sortie du réseau et la sortie désirée :

$$\boldsymbol{E(n) = f(y_{ik})} \qquad \textbf{(3)}$$

Où $y_L(n)$ est la sortie du réseau au temps n et (n) la sortie désirée. L'algorithme de BP assure la descente du gradient sur le critère d'erreur pour atteindre le minimum. Le gradient (n) d'est calculé pour tous les poids de la manière suivante :

$$\frac{\partial \mathbf{E(n)}}{\partial \boldsymbol{W_{ijK}}} = \frac{\partial \mathbf{E(n)}}{\partial \boldsymbol{y_{ik}}} . \frac{\partial \boldsymbol{y_{ik}}}{\partial \boldsymbol{W_{ijK}}} = \frac{\partial \mathbf{E(n)}}{\partial \boldsymbol{y_{ik}}} . \boldsymbol{X_{i-1j}} \qquad \textbf{(4)}$$

Dans le cas de la couche de sortie (i = L) on évalue le terme d'erreur de sortie δ_{LK} comme suite :

$$\boldsymbol{\delta_{lk}} = \frac{\partial \mathbf{E(n)}}{\partial \boldsymbol{y_{ik}}} = \boldsymbol{2f'(\delta_{lk})(d_k - x_{L,x})} \quad \textbf{(5)}$$

Pour les couches cachées, le terme δ_{lk} d'erreur du neurone (i, k) est donné par:

$$\boldsymbol{\delta_{lk}} = \mathbf{f'(y_{ik})} \sum_{\boldsymbol{j=1}}^{\boldsymbol{N(j+1)}} \boldsymbol{\delta_{lk} * W_{i+kj}} \quad \textbf{(6)}$$

La modification des poids et biais est obtenue selon les équations suivantes :

$$\boldsymbol{w_{ijk}(n+1) = w_{ijk}(n) + \eta^{7} \delta_{ik} x_{x-j} + \Omega^{8}(w_{ijk}(n) - w_{ijk}(n-1))} \qquad \textbf{(8)}$$

[7] Le taux d'apprentissage

[8] Ce paramètre permet l'introduction de l'ancien poids dans le calcul de sa nouvelle valeur

Chapitre 4 : Travail Réalisé

I. Introduction

La reconnaissance automatique des caractères et des chiffres dans des images bruité est l'objectif primordial de notre projet, et pour cela on a utilisé un outil qui entre dans le cadre de 'intelligence artificiels, c'est les réseaux de neurones artificiels.

Apres avoir décrits les réseaux de neurones et le problème de la reconnaissance des lettres et des chiffres manuscrite dans les 2 chapitres précédents, 3 points essentiels seront réaliser dans ce chapitre.

- La création du réseau.
- L'entrainement et le teste du réseau.
- Utilisation définitive du réseau.

II. Configuration matériel

Pour la Reconnaissance des caractères manuscrits, nous avons utilisé un pc portable personnel possédant cette configuration :

- Processeur Intel(R) Core™ i7-8565U CPU @ 1.80GHZ 1.99GHZ
- Mémoire vive de 8 Go
- Disque dur hybride SSD de capacité 256 Go
- Système d'exploitation Windows 10 x64 bits

III. Démonstration

1. Création du Réseaux de neurones

On a fait un programme qui permet la création des Réseaux de neurones d'une façon générale qui nous permet de définir les paramètres du Réseau qui définissent l'architecture du Réseau quel que soit le nombre d'entrées et le nombre de sortie et le nombre de couche cachée.et pour cela on a créé quatre Classes :

Classe Neurone :

On a défini cette classe pour initialiser un neurone, et pour cela on a déclaré Un tableau de réelle pour stocker les valeurs d'entrés du neurone et un autre tableau de même type pour stocker les valeurs du poids. Et deux réelles, l'une pour stocker la valeur du biais et l'autre pour stocker la valeur du sortie.

Aussi on a défini une fonction qui permet d'initialiser les valeurs du poids et du biais d'une manière aléatoire, et la fonction d'activation qui permet de calculer la valeur de sortie (fonction sigmoïde).

Classe Réseau :

Pour la création du réseau on a défini une classe qui permet la création d'un réseau quel que soit son architecture et pour cela on a déclaré deux tables entiers qui indique le nombre du valeurs d'entrées et de sorties , aussi un tableau de plusieurs dimensions dont le nombre de colonnes indique le nombre de couche cachée et la valeur de chaque colonne indique le nombre de neurones dans la couche cachée i. et un vecteur qui représente le chiffre correspondant pour chaque image avec toutes les valeurs à 0, sauf pour le chiffre correct. Par

exemple, si l'image contient le chiffre a, alors le vecteur de « labels » pour cette image correspond à : [1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0] parce que chaque indice du tableau correspond au chiffre que nous essayons de reconnaitre comme des attributs .

Pour le taux d'apprentissage on a utilisé la valeur 0.3. Le taux d'apprentissage est un paramètre qui permet d'indiquer à quelle vitesse les poids du réseau seront modifiés. Si cette valeur est trop petite, il faut énormément de données pour que le réseau apprenne correctement. A l'inverse, si la valeur est trop élevée, le réseau apprend plus vite mais il ne sera peut-être pas assez précis car les adaptations des poids seront trop grandes pour qu'elles se rapprochent de la bonne valeur.

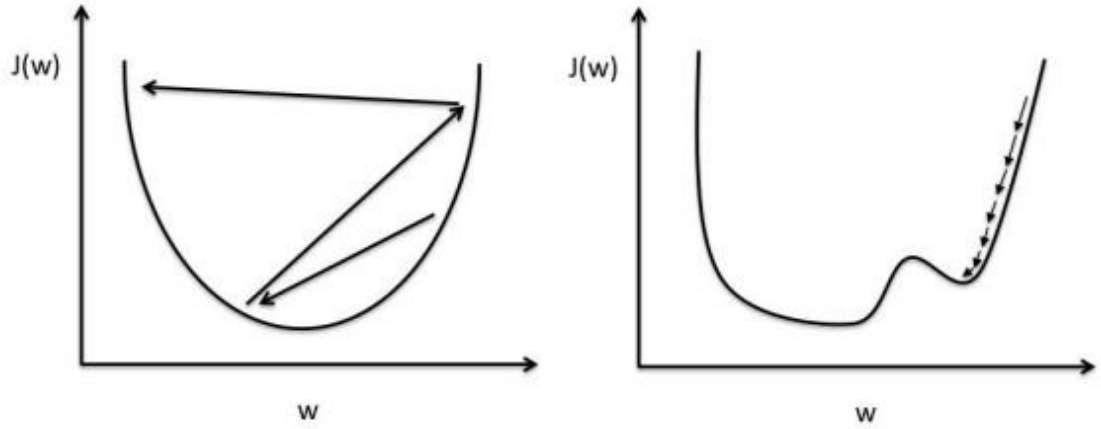

Figure 15: Fonction de cout dans un réseau de neurones artificiels

Pour chaque itération d'apprentissage nous calculons une fonction de coût - Figure 31. Cette fonction de coût représente la différence entre le résultat obtenu et le résultat attendu. Lors de l'apprentissage, nous essayons de trouver les poids qui nous donnent la valeur la plus faible dans notre fonction de coût. La fonction doit déterminer comment les poids seront adaptés. Le taux d'Apprentissage définit à quelle vitesse les poids changeront. A gauche, nous pouvons voir que les changements sont trop importants pour atteindre un résultat satisfaisant. Cependant, à droite, le taux d'Apprentissage fait qu'il est nécessaire d'avoir plus d'itérations d'apprentissage, mais les changements sont assez petits pour atteindre un coût minimal**.**

Pour construire notre Réseaux on définit un constructeur qui permet d'initialiser tous les poids et les bais de notre Réseaux en utilisant la méthode **Math.random()** qui renvoie un nombre de type double aléatoire supérieur ou égal à 0,0 et inférieur à 1,0.

Dans notre application on a utilisé deux réseaux de neurone, un pour la reconnaissance des lettres et l'autre pour la reconnaissance des chiffres.

- Pour le RN concernent la reconnaissance des lettres, nous aurons 784 entrées car chaque image fait 28x28 pixels et 26 sortie car cela correspond au nombre d'alphabet que nous essayons de reconnaitre.
- Pour le RN concernent la reconnaissance des chiffres, nous aurons 784 entrées car chaque image fait 28x28 pixels et 10sortie car cela correspond au nombre de chiffre que nous essayons de reconnaitre

Pour les réseaux que nous avons créé pour les reconnaissances on a défini 3 couches cachées (500 neurones, 300 neurones, 100 neurones).

2. L'entrainement du réseau de neurones

Pour l'entrainement des réseaux on a utilisé les deux bases de données MNIST et EMNIST **(voir chapitre 1).**

On a obtenu ces deux bases de donnes dans le site web **[14]** comme un fichier csv, et pour lire les données nous avons défini une classe **DonnesAPP**. Cette classe permet de lire les données et de remplir une matrice dont chaque ligne contient les valeurs d'entrées et la valeur de sortit, c'est laquelle on doit prédire.

Pour trainer notre Réseau on a utilisé une technique d'apprentissage supervisé appelée rétropropagation. La rétropropagation est une méthode utilisée dans les réseaux de neurones artificiels pour calculer le gradient nécessaire au calcul des poids à utiliser dans le réseau **(voir le chapitre 4).** Dans le contexte de l'apprentissage, l'algorithme d'optimisation de la descente du gradient utilise la rétropropagation pour ajuster le poids des neurones en calculant le gradient de la fonction de cout.

Pour applique la méthode de Rétropropagation de gradient on a défini 2 fonction :

- **La fonction de propagation :**

C'est une fonction qui permet de reliée tous les neurones du Réseaux, de façon générale, un neurone collecte des contributions activatrices et inhibitrices et y ajoute un biais, puis calcule un niveau d'activation en fonction de son résultat et d'une fonction d'activation.

- **La fonction de retropropagation :**

C'est une fonction qui permet d'appliquer l'algorithme de retropropagation de gradient **(voir chapitre 4)**

Pour l'entrainement on a divisé les données d'apprentissages en trois parties, et à chaque fois on traine par deux parties et la partie qui reste c'est pour tester notre réseau. On le fait 3 fois successive.

Chaque entrainement nous a pris 309 min **(voir figure 16).**

```
------------ITERATION N° 91----------
Erreur: 1.4568340832703812  Temps: 146,471 s
Temps Iteration: 146,471 s
------------ITERATION N° 92----------
Erreur: 1.5005520215540666  Temps: 131,182 s
Temps Iteration: 131,182 s
------------ITERATION N° 93----------
Erreur: 1.4671565121069592  Temps: 138,554 s
Temps Iteration: 138,554 s
------------ITERATION N° 94----------
Erreur: 1.4579039585501927  Temps: 156,339 s
Temps Iteration: 156,339 s
------------ITERATION N° 95----------
Erreur: 1.3333468745866728  Temps: 143,197 s
Temps Iteration: 143,197 s
------------ITERATION N° 96----------
Erreur: 1.4053067695202497  Temps: 130,974 s
Temps Iteration: 130,974 s
------------ITERATION N° 97----------
Erreur: 1.3996241179597444  Temps: 135,143 s
Temps Iteration: 135,143 s
------------ITERATION N° 98----------
Erreur: 1.4816778825640664  Temps: 151,704 s
Temps Iteration: 151,704 s
------------ITERATION N° 99----------
Erreur: 1.6290540085269225  Temps: 142,05 s
Temps Iteration: 142,05 s
Temps Apprantissage: 309,576 min
```

Figure 16:entrainement du Réseau

3. Test de performance

Une fois le réseau est entrainé, nous pouvons l'utiliser pour un groupe de test indépendant des groupes d'apprentissage et de validation, et pour cela on a stocké les valeurs du poids dans un fichier pour l'utiliser lors du Test.

```
Nombre des Neurones par couches: 784,500,300,100,26
Couche N° 0:
Neurone N° 0: 0.9816598742565621,0.5968214904248454,0.5144084119644237,0.28665059679716426,-0.475
Neurone N° 1: 0.7784442748311522,0.49940217275430676,-0.746144343790885,0.5037159832676912,0.8526
Neurone N° 2: 0.5022461458106449,-0.3168979239062679,0.4901751880265055,0.08979564976651022,0.020
Neurone N° 3: -0.9127545432686697,0.5732282646790898,-0.5599360414881969,-0.24694975206813133,0.4
Neurone N° 4: -0.7046569003991514,-0.5281303780160946,-0.055909830481080336,-0.18128644129447588,
Neurone N° 5: -0.5489144750581942,0.13120042480153593,-0.6802183052572812,-0.2903605895095619,0.6
Neurone N° 6: 0.4344014472151134,-0.602904392341119,0.5975271308371781,0.6222114724628804,-0.0790
Neurone N° 7: 0.9311511113613729,-0.07694062840957727,-0.8607111323781436,0.39714296120899806,-0.
Neurone N° 8: -0.05851186463853564,0.9544656024529923,0.6401292651614356,-0.41334965212922686,0.7
Neurone N° 9: -0.4539661456894648,0.18311139518228603,-0.48745117986875686,-0.28454471392310626,0
Neurone N° 10: 0.929864151104226,-0.32791746339827865,0.6536372006372693,0.7385355163582882,-0.10
Neurone N° 11: 0.6643086694290072,0.8509317663474938,-0.40496349125925235,0.6929302049802045,0.05
Neurone N° 12: 0.24656278158565326,-0.41883028839828773,0.03571723262329263,0.5458902571570555,-0
Neurone N° 13: -0.10288821039478235,0.29206773567438704,-0.2424334841364313,-0.7622235630085376,-
Neurone N° 14: -0.38503220820882356,0.18677399683529705,-0.07621073463387162,-0.0796068044020553,
Neurone N° 15: 0.4907188719610396,0.5187415700063238,0.7601212783785585,0.39061649267134213,0.727
Neurone N° 16: -0.7226087180603202,-0.061531025959041585,0.6188551783584733,0.7194047192172066,-0
Neurone N° 17: -0.7808767387012894,0.9105538049455604,-0.5743210042224165,-0.923936659078295,-0.4
Neurone N° 18: 0.13765220797545386,0.2010579998274058,-0.4906732468870898,0.7478563280704328,-0.0
Neurone N° 19: 0.7766138735681449,0.43518641540074565,0.8140420226006211,-0.6451853101446227,0.60
Neurone N° 20: 0.4036542240573133,-0.27472272436176803,-0.5051461839974445,-0.7398542395423127,0.
```

Figure 17: fichier qui stocke les valeurs du poids pour la RDCA[9]

[9] Reconnaissances des caractères

```
Nombre des Neurones par couches: 784,500,300,100,10
Couche N° 0:
Neurone N° 0: -0.25384261997301616,0.4313785363564069,-0.2505957464803661,0.3028807729796823,-0.2026875433248
Neurone N° 1: 0.9914266902347653,-0.20926718175147974,-0.8021000858650604,-0.15996028982869626,-0.07419380658
Neurone N° 2: -0.06673146715492595,-0.9019554481843013,-0.4457010660060534,0.2904760810343756,-0.261866278087
Neurone N° 3: -0.90187485737489,0.9078829004878572,-0.7312326600363064,0.30757209937032814,-0.603583367301969
Neurone N° 4: -0.4289898631994824,-0.6325655009469009,0.11714001917592576,0.45209349566261636,-0.760350709918
Neurone N° 5: -0.6738641421521436,-0.6613652373386494,-0.3190379682742748,0.5027972700259009,0.55377934315506
Neurone N° 6: -0.9346465400556645,-0.15938796272586453,-0.5396570703396222,0.21235226611227187,0.099600879884
Neurone N° 7: 0.5644052175341783,0.8124434739733972,-0.38037706513093905,0.5085718954222611,-0.60154023767090
Neurone N° 8: 0.5269894583429646,0.19622431096058612,-0.6341477873302652,0.1409597719798632,-0.22422806722879
Neurone N° 9: 0.94663810593879,0.3404489390194414,-0.039917787086887646,0.7725674378077017,0.1623539605797259
Neurone N° 10: 0.976406548718183,0.7237048382272333,-0.6385303104486832,-0.48283518004773107,0.65356593413079
Neurone N° 11: -0.6936238219565196,-0.0013210164168848681,-0.49580670563689955,-0.07827680445817631,0.8262671
Neurone N° 12: 0.0489691335956739,0.3564766450477783,-0.7928175084841174,0.394210560568818,-0.186415675304163
Neurone N° 13: -0.8679240139003845,-0.7887473136974616,-0.2833409537156528,-0.9233004634926965,-0.86250824757
Neurone N° 14: 0.19999544845075734,-0.7948445036616967,-0.11850880776627015,0.16378777006183265,-0.7007952229
Neurone N° 15: -0.9778129757857448,-0.46641666964299233,-0.03864007447517248,0.6663309303559866,-0.8726147900
Neurone N° 16: -0.5901642558697533,-0.4092469847179754,-0.9322618733046897,0.6204879012291717,0.8841593006470
Neurone N° 17: -0.359770130206182,-0.0644184883841199,-0.7623179520186658,-0.25792095218979627,0.128948474162
Neurone N° 18: 0.3889505065272101,-0.4221203389069865,-0.8595647295124118,-0.4737437390333552,-0.659887734818
Neurone N° 19: -0.8301256377593522,0.584858462854291,0.41684526531622224,-0.2075139537713946,0.29440399912419
Neurone N° 20: 0.282988475090054,-0.9920519877462761,-0.34840757545579426,0.6748449866347499,0.20423866942014
Neurone N° 21: -0.7299822705044834,-0.8700710914177032,0.9782314577409923,-0.9423724652029732,0.6127737707592
Neurone N° 22: 0.8778647879914778,0.03237587838916345,0.3170454371485478,0.4562664769108278,0.046802149565229
Neurone N° 23: -0.47013739717858116,-0.5165855958382446,0.2457124468202625,-0.6564342144114841,-0.57254236834
Neurone N° 24: -0.6951608056473455,0.14578927058003543,0.5027842843757797,0.9212461446383824,-0.7612668560353
Neurone N° 25: 0.4239953072588478,-0.8947537050564032,-0.8760216670411654,-0.6631972267399939,-0.546188744177
Neurone N° 26: -0.7544059346982666,-0.2807742632456711,-0.7887188420145377,-0.6543241672219267,-0.78079287510
Neurone N° 27: 0.7674638019037834,0.7858690404835416,0.33486542310682776,0.41873340101350354,-0.7032757201087
Neurone N° 28: 0.637312561494537,0.033337376703943544,-0.43218137963060665,0.29506682057677636,0.263222480151
Neurone N° 29: -0.32914404453819324,0.9294431587782113,-0.8463153870497728,0.980098584996244,-0.5318015284350
```

Figure 18: fichier qui stocke les valeurs du poids pour la RDCI[10]

La précision du notre réseau (la reconnaissance des caractères) est 86%

[10] Reconnaissance des chiffres

```
Test N° 22774: 7===7 Temps: 0,001 s
Test N° 22775: 19===19 Temps: 0 s
Test N° 22776: 7===7 Temps: 0,001 s
Test N° 22777: 15===15 Temps: 0 s
Test N° 22778: 4===15 Temps: 0,001 s
Test N° 22779: 12===12 Temps: 0 s
Test N° 22780: 11===24 Temps: 0 s
Test N° 22781: 3===3 Temps: 0,001 s
Test N° 22782: 20===20 Temps: 0,001 s
Test N° 22783: 17===7 Temps: 0 s
Test N° 22784: 18===18 Temps: 0,001 s
Test N° 22785: 7===7 Temps: 0 s
Test N° 22786: 15===15 Temps: 0 s
Test N° 22787: 4===4 Temps: 0,001 s
Test N° 22788: 4===4 Temps: 0 s
Test N° 22789: 21===21 Temps: 0,001 s
Test N° 22790: 14===14 Temps: 0 s
Test N° 22791: 8===8 Temps: 0,001 s
Test N° 22792: 6===6 Temps: 0 s
Test N° 22793: 17===17 Temps: 0,001 s
Test N° 22794: 19===19 Temps: 0 s
Test N° 22795: 1===1 Temps: 0,001 s
Test N° 22796: 21===21 Temps: 0 s
Test N° 22797: 1===18 Temps: 0,001 s
Test N° 22798: 23===23 Temps: 0,001 s
Test N° 22799: 12===9 Temps: 0 s
-----TEST FINIE : 19821/22800 PRECISION : 86.9342105263158% TEMPS TEST: 0,173 min
```

Figure 19: Test de précision.

IV. Description de l'application

Dans cette partie nous présentons les interfaces de l'application « hcrAPP[11]» ainsi les scenarios d'exécution pour montrer quelques résultats de tests de notre application.

Nous avons utilisé JavaFx. JavaFX est une plate-forme d'applications clientes open source de nouvelle génération pour les systèmes de bureau, mobiles et embarqués basés sur Java. Il s'agit d'un effort de collaboration de nombreuses personnes et entreprises dans le but de produire une boîte à outils moderne, efficace et complète pour le développement d'applications clientes riches.

[11] handwritten character recognition Application

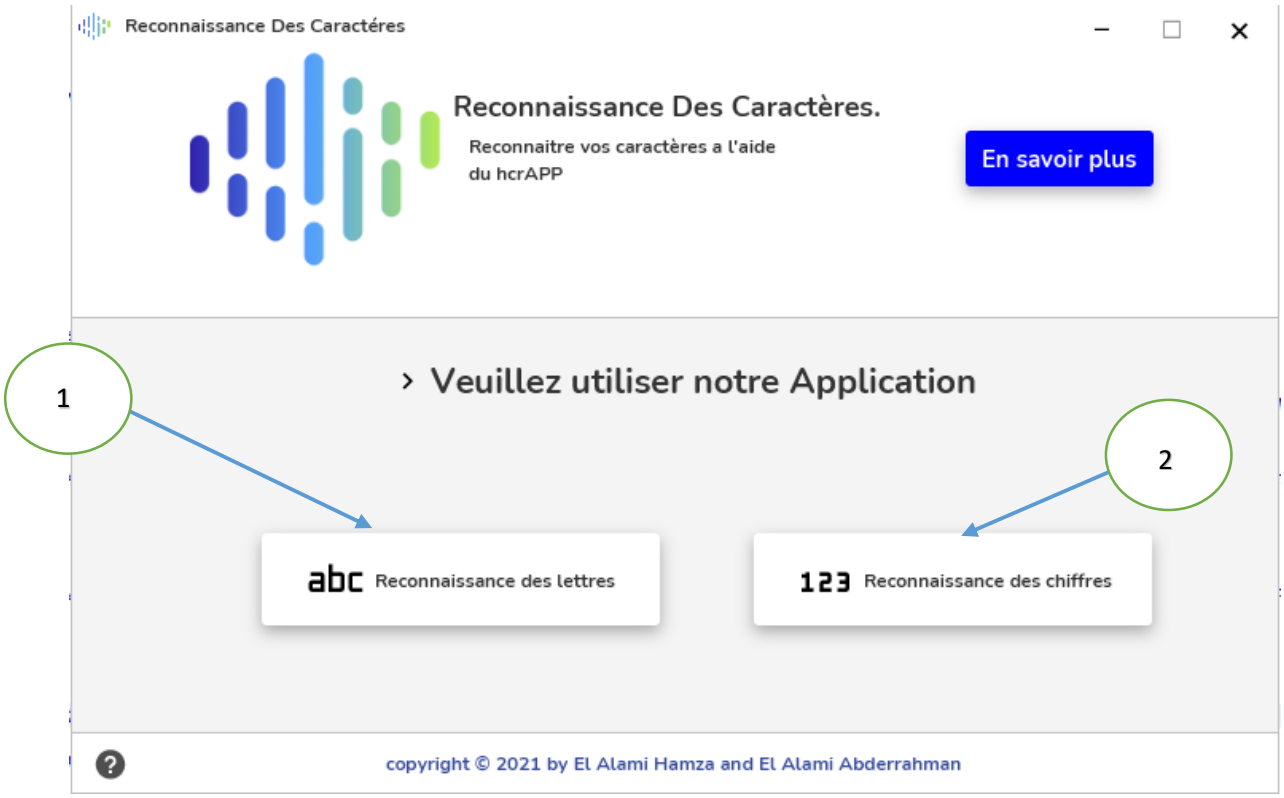

Figure 20:application

Le bouton **« en savoir plus »** donne une brève explication sur les réseaux de neurones artificiels.

Les principaux boutons de notre Application c'est les boutons 1 et 2, ils ont le même fonctionnement sauf que le premier bouton est designer pour la reconnaissance des caractères et le deuxième pour la reconnaissance des chiffres.

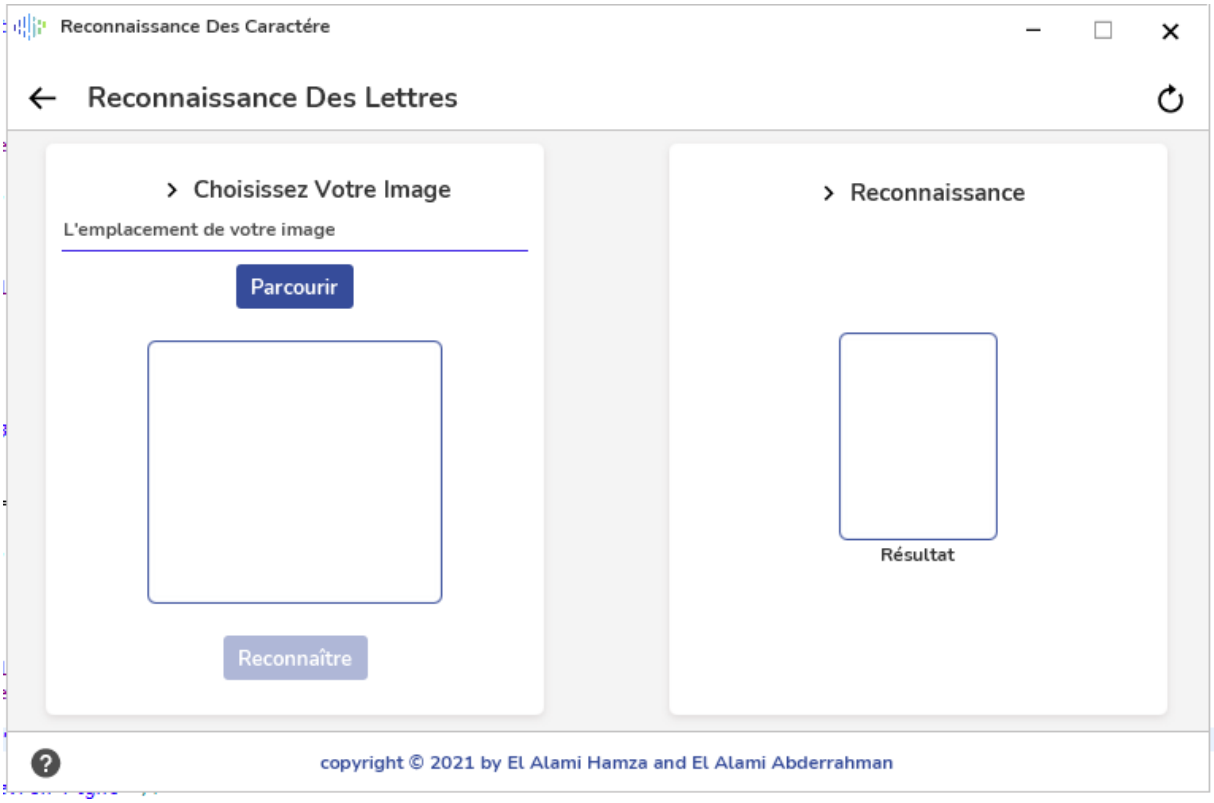

Figure 21: Application 2.

Le bouton parcourir permet de choisir l'image qui contient le caractère manuscrit, quand l'mage est bien choisir le bouton prédire activer et quand on clique sur ce bouton, le programme transforme l'mage choisit au tableau de taille 784 dont chaque colonne contient la valeur du pixel de la position i et on stocke le résultat dans un fichier CSV et ensuite l'application propage cette résultat dans notre réseau qui est déjà entrainer. Enfin, Le résultat du résultat sera affiché dans la zone de Résultats.

V. Test de L'Application

Une fois le réseau est entrainé, nous pouvons l'utiliser pour un groupe de test indépendant des groupes d'apprentissage et de validation. Et pour cela on décrit une fonction qui permet la conversion d'image au tableau de 784 colonnes, dont chaque colonnes décrit la valeur du pixel de 0 à 255 **(voir chapitre 2).**

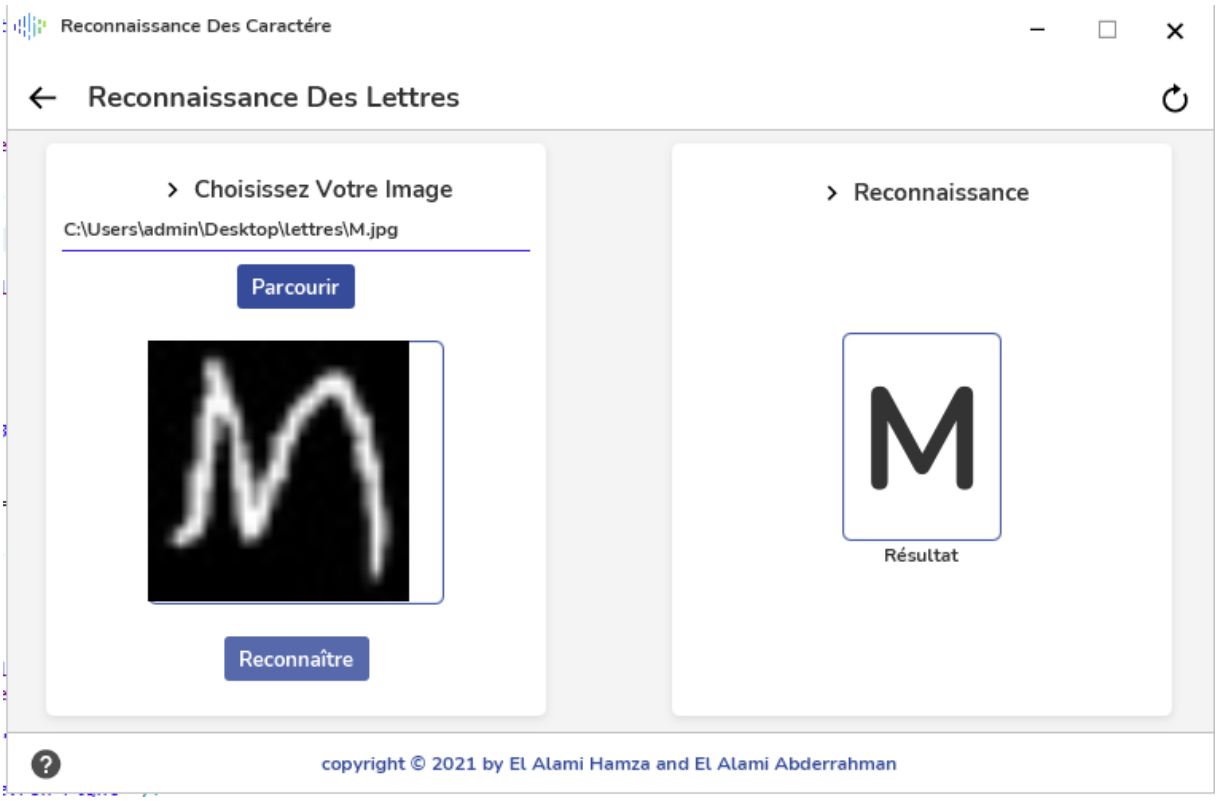

Figure 22: test d'Application

Conclusion Générale

La classification d'images est une tâche importante dans le domaine de la vision par ordinateur, la reconnaissance d'objets et l'apprentissage automatique. Grâce à l'apprentissage automatique, l'avenir de l'IA se développe très rapidement. Dans ce projet nous avons entamé les notions fondamentales sur les réseaux de neurones ainsi on a montré comment utiliser cette dernière pour la reconnaissance des caractères manuscrites. Les résultats obtenus dans ce Projet ont montré que le réseau de neurones est un puissant outil de classification des données avec des taux d'erreur très petits.

La réalisation de ce projet nous a permet de bien comprendre l'Intelligence artificiel, en particulier les réseaux de neurones, de comprendre et d'assimiler les notions de base du RNA, et dans le futur, nous aimerons bien approfondir les connaissances de l'IA[12]. De bien comprendre et manipuler les CNN[13]. *Les réseaux convolutifs sont une forme particulière de RNN dont l'architecture des connexions est inspirée de celle du cortex visuel des mammifères* ***(voir chapitre 4).***

Pour finir, La réalisation de ce Projet a été une expérience enrichissante et motivantes. En effet, nous aimerons bien suivre un Master en Sciences de l'Information dans lequel les concepts de l'intelligence artificiels seront abordés.

[12] Intelligence artificiels

[13] Réseaux de neurones convulitifs

Annexe

Classe de Neurone :

```
public class Neurone {

	private double[] valeursEntres;
	private double[] valeursPoids;
	private double biais;
	private double sortie;
		// POUR LA RETROPROPAGATION
	private double[] retroPoids;
	private double erreur;

		// CONSTRUCTEURS
	public Neurone(int nbrValeursEntres,int nbrRetroEntres) {
		this.valeursEntres= new double[nbrValeursEntres];
		this.valeursPoids= new double[nbrValeursEntres];
		this.retroPoids= new double[nbrRetroEntres];
		this.erreur= 0;
		this.biais= ReseauAide.randomValeur(-0.5,0.7);
	}
	public Neurone(int nbrValeursEntres) {
		this.valeursEntres= new double[nbrValeursEntres];
		this.valeursPoids= new double[nbrValeursEntres];
		this.erreur= 0;
		this.biais= ReseauAide.randomValeur(-0.5,0.7);
	}
		// GETTERS
	public double[] getEntres() {return valeursEntres;}
	public double[] getPoids() {return valeursPoids;}
	public double getBiais() {return biais;}
	public double[] getRetroPoids() {return retroPoids;}
	public double getErreur() {return erreur;}
	public double getSortie() {return sortie;}

		// SETTERS
	public void setBiais(double biais) {this.biais = biais;}
	public void setErreur(double erreur) {this.erreur = erreur;}
	public void setSortie(double sortie) {this.sortie = sortie;}

		// LA SOMME DES POIDS*ENTRES
	public double fctAffine() {
		double somme= this.biais;
		for(int i=0;i<this.valeursEntres.length;i++)
			somme= somme+(this.valeursEntres[i]*this.valeursPoids[i]);
		return somme;
	}

		// LA FONCTION D'ACTIVATION
		// SIGMA
	public double sigma(double fctAffine) {
		return 1d/(1+Math.exp(-fctAffine));
	}
		// LA DERIVEE DE SIGMA
	public double derSigma(double erreur) {
		return sigma(erreur) *(1-sigma(erreur));
	}

		// LA SORTIE DU NAURONE
	public double sortie() {
```

```
		sortie= sigma(fctAffine());
		return sigma(fctAffine());
	}

		// LA SORTIE DE LA RETROPRAGATION DU NAURONE
	public double sortieRetro() {
		return derSigma(this.fctAffine());
	}
		// INITIALISER LES POIDS DU NAURONE
	public void initPoidsNaurone() {
		for(int i= 0;i<this.valeursPoids.length;i++)
			this.valeursPoids[i]= ReseauAide.randomValeur(-1,1);
	}

		// INITAILISER LES ENTRES DES NAURONE
	public void initEntresNaurone(double[] entres) {
		for(int i= 0;i<this.valeursEntres.length;i++)
			this.valeursEntres[i]= entres[i];
	}

		// AFFICHAGE DES POIDS ET LE BIAIS
	public void afficher(int numNaurone) {
		System.out.print("Naurone "+numNaurone+": [");
		for(int i=0;i<this.valeursPoids.length;i++)
			System.out.print("<"+this.valeursPoids[i]+">,");
		System.out.println("]|[<"+this.biais+">]");
	}
}
```

Classe de Reseau :

```
public class Reseau {
	private final String NOM_RESEAU;
	private final int[] NBR_NAURONES_COUCHE;
	private final int TAILLE_ENTRES;
	private final int TAILLE_SORTIES;
	private final int TAILLE_RESEAU;
	private final double RATE_APP=0.3;
	private Neurone[][] neurones;
		// POUR LA RETROPROPAGATION
	public double[] sortiesProduit;

		// CONSTRUCTEURS
	public Reseau(String NOM_RESEAU,int nbrEntres,int... NBR_NAURONES_COUCHE) {
		this.NOM_RESEAU= NOM_RESEAU;
		this.NBR_NAURONES_COUCHE= NBR_NAURONES_COUCHE;
		this.TAILLE_RESEAU= NBR_NAURONES_COUCHE.length;
		this.TAILLE_ENTRES= nbrEntres;
		this.TAILLE_SORTIES= NBR_NAURONES_COUCHE[this.TAILLE_RESEAU-1];
		this.sortiesProduit= new double[this.TAILLE_SORTIES];
		this.neurones= new Neurone[this.TAILLE_RESEAU][];
		int nbrEntre= this.TAILLE_ENTRES; int nbrEntresRetro= 0;  /// AIDE
!!!!!
		for(int i=0;i<this.TAILLE_RESEAU;i++) {
			if(i<this.TAILLE_RESEAU-1)
				nbrEntresRetro= this.NBR_NAURONES_COUCHE[i+1];
			neurones[i]= new Neurone[this.NBR_NAURONES_COUCHE[i]];
			for(int j= 0;j<this.NBR_NAURONES_COUCHE[i];j++) {
				if(i<this.TAILLE_RESEAU-1)
					neurones[i][j]= new
Neurone(nbrEntre,nbrEntresRetro);
				else
```

```
                                neurones[i][j]= new Neurone(nbrEntre);
                        }
                        nbrEntre= this.NBR_NAURONES_COUCHE[i];
                }
        }
        public Reseau(int nbrEntres,int... NBR_NAURONES_COUCHE) {
                this.NOM_RESEAU = "";
                this.NBR_NAURONES_COUCHE= NBR_NAURONES_COUCHE;
                this.TAILLE_RESEAU= NBR_NAURONES_COUCHE.length;
                this.TAILLE_ENTRES= nbrEntres;
                this.TAILLE_SORTIES= NBR_NAURONES_COUCHE[this.TAILLE_RESEAU-1];
                this.sortiesProduit= new double[this.TAILLE_SORTIES];
                this.neurones= new Neurone[this.TAILLE_RESEAU][];
                int nbrEntresAide= nbrEntres;  /// AIDE !!!!!
                for(int i=0;i<this.TAILLE_RESEAU;i++) {
                        neurones[i]= new Neurone[this.NBR_NAURONES_COUCHE[i]];
                        for(int j= 0;j<this.NBR_NAURONES_COUCHE[i];j++)
                                neurones[i][j]= new Neurone(nbrEntresAide);
                        nbrEntresAide= this.NBR_NAURONES_COUCHE[i];
                }
        }

                // INITIALISATION DES POIDS
        public void initPoidsReseau() {
                for(int i= 0;i<this.TAILLE_RESEAU;i++) {
                        for(int j= 0;j<this.NBR_NAURONES_COUCHE[i];j++)
                                this.neurones[i][j].initPoidsNeurone();
                }
        }
        public void initPoidsRetroReseau() {
                for(int i= 0;i<this.TAILLE_RESEAU-1;i++) {
                        for(int j= 0;j<this.NBR_NAURONES_COUCHE[i];j++)
                                for(int k= 0;k<this.NBR_NAURONES_COUCHE[i+1];k++)
                                        this.neurones[i][j].getRetroPoids()[k]=
this.neurones[i+1][k].getPoids()[j];
                }
        }

                // LA PROPAGATION
        public void propagation(double[] entres) {
                double[] entresAide= entres;
                double[] sortie;
                for(int i= 0;i<this.TAILLE_RESEAU;i++) {
                        sortie= new double[NBR_NAURONES_COUCHE[i]];
                        for(int j= 0;j<NBR_NAURONES_COUCHE[i];j++) {
                                this.neurones[i][j].initEntresNeurone(entresAide);
                                sortie[j]= this.neurones[i][j].sortie();
                                //System.out.println(Arrays.toString(sortie));
                                if(i== this.TAILLE_RESEAU-1) {
                                        this.sortiesProduit[j]=sortie[j];
                                //
        System.out.println(Arrays.toString(sortiesProduit));
                                }
                        }
                        if(i<this.TAILLE_RESEAU-1) {
                                entresAide= sortie;
                        }
                }
        }

                // LA RETOPROPAGATION
        public void retroPropagation(double[] sortie) {
```

```
		int dernierCouche= this.TAILLE_RESEAU-1;
			// INITIALISER LES ERREURS DE LA DERNIER COUCHE
		for(int i= 0;i<this.NBR_NAURONES_COUCHE[dernierCouche];i++)

	this.neurones[dernierCouche][i].setErreur((this.sortiesProduit[i]-
sortie[i])*this.neurones[dernierCouche][i].sortieRetro());
			// INITIALISER LES POIDS DES COUCHES RESTANTES
		for(int i= dernierCouche-1;i>=0;i--) {
			for(int j= 0;j<this.NBR_NAURONES_COUCHE[i];j++)

	this.neurones[i][j].setErreur(this.neurones[i][j].sortieRetro()*this.sommeRe
tro(i,j));
		}
			// CALCULE DES NEUVAUX POIDS ET DES BIAIS
		for(int i= 0;i<this.TAILLE_RESEAU;i++)
			for(int j= 0;j<this.NBR_NAURONES_COUCHE[i];j++) {
				for(int k=
0;k<this.neurones[i][j].getPoids().length;k++) {
					this.neurones[i][j].getPoids()[k]=
this.neurones[i][j].getPoids()[k]-
(this.RATE_APP*this.neurones[i][j].getEntres()[k]*this.neurones[i][j].getErreur())
;
				}

	this.neurones[i][j].setBiais(this.neurones[i][j].getBiais()-
(this.RATE_APP*this.neurones[i][j].getErreur()));
			}
			// INITIALISER LES POIDS RETRO
		this.initPoidsRetroReseau();
	}

	public double sommeRetro(int posCouche,int posNaurone) {
		double somme= 0;
		for(int i= 0;i<this.NBR_NAURONES_COUCHE[posCouche+1];i++)
			somme=
somme+(this.neurones[posCouche][posNaurone].getRetroPoids()[i]*this.neurones[posCo
uche+1][i].getErreur());
		return somme;
	}

	public void apprantissage(DonneesApp TS,int nbrIterations,int nbrLoops,int
nbrDonneesUtiliser) {
		for(int j= 0;j<nbrIterations;j++) {
			System.out.println("------------ITERATION N° "+j+"----------");
			for(int i= 0;i<nbrLoops;i++) {
				DonneesApp batch =
TS.donneesAUtiliser(nbrDonneesUtiliser);
//				System.out.println(batch);
				for(int k= 0;k<nbrDonneesUtiliser;k++) {
					this.propagation(batch.getEntres(k));
					this.retroPropagation(batch.getSorties(k));
				}
				System.out.println(this.erreurTotale(batch));
			}
		}
	}

	public void testReseau(DonneesApp DA) {
		int correcte= 0;
		int indiceSortieProduit, indiceSortieDesirer;
		for(int i= 0;i<DA.taille();i++) {
			this.propagation(DA.getEntres(i));
```

```
                        indiceSortieProduit=
ReseauAide.indiceValeurMax(this.sortiesProduit)+1;
                        indiceSortieDesirer=
ReseauAide.indiceValeurMax(DA.getSorties(i))+1;
                        System.out.println("test n° "+i+"--
"+indiceSortieDesirer+"==="+indiceSortieProduit+"
"+ReseauAide.valeurMax(sortiesProduit)+"%");
                        if(indiceSortieProduit== indiceSortieDesirer)
                              correcte++;
                  }
                  System.out.println("-----TEST FINIE : "+correcte+"/"+DA.taille()+"
PRECISION : "+(double)correcte/DA.taille()*100+"%");
            }

            public  double erreurTotale(DonneesApp DA) {
                  double erreurTotale= 0;
                  double erreur;
                  for(int i= 0;i<DA.taille();i++) {
                        this.propagation(DA.getEntres(i));
                        erreur= 0;
                        for(int j= 0;j<DA.getSorties(0).length;j++) {
                              erreur+= Math.pow(DA.getSorties(i)[j]-
this.sortiesProduit[j],2);
                        }
                        erreur= erreur/ 2d* DA.getSorties(i).length;
                        erreurTotale= erreurTotale+erreur;
                  }
                  erreurTotale= erreurTotale/ DA.taille();
                  return erreurTotale;
            }

                  // STOCKAGE DES POIDS ET DES BIAIS D'UN RESEAU DANS UN FICHIER
            public void enregistrerReseau(File fichier) throws IOException {
                  FileWriter fw=new FileWriter(fichier);
                  fw.write("Nom du Reseau: "+this.NOM_RESEAU+"\n");
                  fw.write("Nombre des Neurones par couches: "+this.TAILLE_ENTRES+",");
                  for(int i= 0;i<this.TAILLE_RESEAU;i++) {
                        fw.write(this.NBR_NAURONES_COUCHE[i]+"");
                        if(i!= this.TAILLE_RESEAU-1)
                              fw.write(",");
                        else
                              fw.write("\n");
                  }
                  for(int i= 0;i<this.TAILLE_RESEAU;i++) {
                        fw.write("Couche N° "+i+":\n");
                        for(int j= 0;j<this.NBR_NAURONES_COUCHE[i];j++) {
                              fw.write("Neurone N° "+j+": ");
                              for(int k=
0;k<this.neurones[i][j].getPoids().length;k++) {
                                    fw.write(this.neurones[i][j].getPoids()[k]+",");
                              }
                              fw.write(this.neurones[i][j].getBiais()+"\n");
                        }
                  }
                  fw.close();
                  System.err.println("------------Enregistrement Términer------------
");
            }

                  // UTILISATION D'UN RESEAU STOCKER DANS LA BASE DE DONNEE
            public static Reseau getReseau(File fichier) throws IOException  {
                  String contenueLigne="";
```

```
            BufferedReader lire= new BufferedReader(new FileReader(fichier));
            String nomReseau= lire.readLine();
            nomReseau= nomReseau.substring(nomReseau.lastIndexOf(":")+2);
            String nbrNeuronesCouche= lire.readLine();
            nbrNeuronesCouche=
nbrNeuronesCouche.substring(nbrNeuronesCouche.lastIndexOf(":")+2);
            String[] aide= nbrNeuronesCouche.split(",");
            int nbrEntres= Integer.parseInt(aide[0]);
            int[] NBR_NEURONES_COUCHE= new int[aide.length-1];
            for(int i= 0;i<aide.length-1;i++)
                  NBR_NEURONES_COUCHE[i]= Integer.parseInt(aide[i+1]);
            Reseau R= new Reseau(nbrEntres,NBR_NEURONES_COUCHE);
            int numNeurone= -1,numCouche= -1;
            while((contenueLigne= lire.readLine())!= null) {
                  if(contenueLigne.substring(0,1).equals("C")) {
                        numCouche++;
                        numNeurone= -1;
                  }
                  else {
                        numNeurone++;
                        String[] aideN=
contenueLigne.substring(contenueLigne.lastIndexOf(":")+2).split(",");
                        for(int i=
0;i<R.neurones[numCouche][numNeurone].getPoids().length;i++)
                              R.neurones[numCouche][numNeurone].getPoids()[i]=
Double.parseDouble(aideN[i]);

      R.neurones[numCouche][numNeurone].setBiais(Double.parseDouble(aideN[aideN.le
ngth-1]));
                  }
            }
            return R;
      }

            // AFFICHAGE DES COMPOSANTES DU RESEAU
      public void afficher(int numIteration) {
            System.out.println("_____________ Iteration N°
"+(numIteration+1)+" _____________");
            System.out.println("- Couche d'entrée :");
            for(int i=0;i<this.NBR_NAURONES_COUCHE[0];i++)
                  this.neurones[0][i].afficher(i);
            for(int i=1;i<this.TAILLE_RESEAU-1;i++) {
                  System.out.println("- Couche cachée N° "+i+" :");
                  for(int j=0;j<this.NBR_NAURONES_COUCHE[i];j++)
                        this.neurones[i][j].afficher(j);
            }
            if(this.TAILLE_RESEAU!=0) {
                  System.out.println("- Couche de sortie :");
                  for(int i=0;i<this.NBR_NAURONES_COUCHE[this.TAILLE_RESEAU-
1];i++)
                        this.neurones[this.TAILLE_RESEAU-1][i].afficher(i);
            }
      }
}
```

Classe ReseauAide :

```
public class ReseauAide {
      public static double[] creerTableau(int size, double init_value){
        if(size < 1){
```

```
            return null;
        }
        double[] ar = new double[size];
        for(int i = 0; i < size; i++){
            ar[i] = init_value;
        }
        return ar;
    }

    public static double randomValeur(double min,double max){
        return Math.random()*(max-min) + min;
    }

    public static Integer[] randomValues(int lowerBound, int upperBound, int
amount) {
        lowerBound --;
        if(amount > (upperBound-lowerBound)){
            return null;
        }
        Integer[] values = new Integer[amount];
        for(int i = 0; i< amount; i++){
            int n = (int)(Math.random() * (upperBound-lowerBound+1) + lowerBound);
            while(containsValue(values, n)){
                n = (int)(Math.random() * (upperBound-lowerBound+1) + lowerBound);
            }
            values[i] = n;
        }
        return values;
    }

    public static <T extends Comparable<T>> boolean containsValue(T[] ar, T
valeur){
        for(int i = 0; i < ar.length; i++){
            if(ar[i] != null){
                if(valeur.compareTo(ar[i]) == 0){
                    return true;
                }
            }
        }
        return false;
    }

    public static int indiceValeurMax(double[] valeurs){
        int indice = 0;
        for(int i = 1; i < valeurs.length; i++){
            if(valeurs[i]>valeurs[indice]){
                indice= i;
            }
        }
        return indice;
    }

    public static double valeurMax(double[] valeurs) {
      double valeurMax= valeurs[indiceValeurMax(valeurs)];
            return valeurMax*100;
    }
}
```

Classe DonneesApp :

```
public class DonneesApp {
      public final int TAILLE_ENTRES;
```

```
	public final int TAILLE_SORTIE;
		// MATRICE AVEC 2 LIGNES 0: POUR LES ENTRES 1: POUR LES SORTIES
ATTANDUE
	private ArrayList<double[][]> donnees = new ArrayList<>();

		// CONSTRUCTEUR
	public DonneesApp(int TAILLE_ENTRES, int TAILLE_SORTIE) {
		this.TAILLE_ENTRES = TAILLE_ENTRES;
		this.TAILLE_SORTIE = TAILLE_SORTIE;
	}

		// POUR AJOUTER DES MATRICES DE DONNEES DANS LA ArrayList
	public void ajtDonnees(double[] entres, double[] sorties) {
		if(entres.length != TAILLE_ENTRES || sorties.length != TAILLE_SORTIE)
return;
		donnees.add(new double[][]{entres, sorties});
	}

		// LE NOMBRES DES DONNEES A UTILISER DANS CHAQUE ITERATION
	public DonneesApp donneesAUtiliser(int taille) {
		if(taille > 0 && taille <= this.taille()) {
			DonneesApp set = new DonneesApp(TAILLE_ENTRES, TAILLE_SORTIE);
			Integer[] ids = ReseauAide.randomValues(0,this.taille()-
1,taille);
			for(Integer i:ids)
				set.ajtDonnees(this.getEntres(i),this.getSorties(i));
			return set;
		}
		else
			return this;
	}

		// GETTERS
    public int taille() {return donnees.size();}
    public int getTAILLE_ENTRES() {return TAILLE_ENTRES;}
    public int getTAILLE_SORTIE() {return TAILLE_SORTIE;}
    public double[] getEntres(int indice) {
        if(indice >= 0 && indice < taille())
            return donnees.get(indice)[0];
        else return null;
    }
    public double[] getSorties(int indice) {
        if(indice >= 0 && indice < taille())
            return donnees.get(indice)[1];
        else return null;
	}
}
```

Classe CSV :

```
public class Csv {
		// CONVERTIR
	static public void convImageCsv(String source) throws IOException {
		BufferedImage bi = ImageIO.read(new URL(source));
	  PrintWriter pw = new PrintWriter(new File("image.csv"));
		StringBuilder sb = new StringBuilder();
		for( int i=0 ; i<bi.getHeight();i++) {
			for(int j=0 ; j<bi.getWidth();j++) {
				Color c = new Color(bi.getRGB(i, j));
```

```
                        int a = c.getRed();
                        int b = c.getGreen();
                        int eee =c.getBlue();
                        double gr =(double)(a + b + eee)/(3*255);

                        sb.append(gr);
                        if(i==bi.getHeight()-1 && j==bi.getWidth()-1)
                                sb.append("\n");
                        else
                                sb.append(",");
                    }
                }
                pw.write(sb.toString());
                pw.close();
        }
                // GET DATA APPRANTISSAGE
        static public DonneesApp getDonneesApp(int debut,int fin,String source)
throws FileNotFoundException,IOException {
                double[] entres,sorties;
                DonneesApp DA= new DonneesApp(784,26);
                String contenueLigne=" ";
                int testTaille= 0;
                BufferedReader lire= new BufferedReader(new FileReader(source));
                System.out.println("-------------PREPARATION DES DONNEES-------------
-");
                    while((contenueLigne= lire.readLine())!= null &&
testTaille<fin) {
                        if(testTaille>=debut) {
                                String[] aideData= contenueLigne.split(",");
                                entres= new double[aideData.length-1];
                                sorties= new double[26];
                                for(int j= 1;j<aideData.length;j++) {
                                        entres[j-1]=
Double.parseDouble(aideData[j])/ (double)255;
                                        if(j== 1)
                                                sorties[Integer.parseInt(aideData[j-
1])]=1d;
                                }
                                DA.ajtDonnees(entres, sorties);
                                if(testTaille% 500== 0)
                                        System.out.println("-"+testTaille+" donnees
sont preparé.");
                        }
                        testTaille++;
                    }
                return DA ;
        }

                // GET DATA TEST
        public static double[] getDonnee(String source) throws IOException {
                double[] entres;
                BufferedReader lire= new BufferedReader(new FileReader(source));
                String[] aideData= lire.readLine().split(",");
                entres= new double[aideData.length];
                for(int j= 0;j<aideData.length;j++)
                        entres[j]= Double.parseDouble(aideData[j]);
                return entres;
        }
}
```

Bibliographie

[1] D.O. HEBB “The organization of behavior” J. WILEY and Sons, 1949

[2] F. ROSENBLATT “Principles of neurodynamics” Spartan, New York, 1962

[3] Yann LE CUN et al. “Handwritten Digit Recognition : Applications of Neural Network Chips and Automatic Learning” IEEE Communications Magazine, November 1989

[4] R.C. Gonzalez, R.E. Woods, " Digital image processing " , Upper Saddle River, N.J. , Prentice Hall, 2002.

[5] A.Christophe, Transmettre et stocker de l'information `' Caractéristiques d'une image numérique : pixellisation, codage RVB et niveaux de gris, octobre 2011

[8] Jean-Paul Haton, Modèles connexionnistes pour l’intelligence artificielle, 1989.

[9] T. M. Hagan, B. H. Demuth, M. H. Beale, O. De Jesús " Neural Network Design ", 2nd Edition, septembre 2014.

[10] S. Haykin, " Neural Networks : A Comprehensive Foundation " , Macmillan College Publishng Company, New York, 1994.

[11] N. Zaarour, " Modélisation d’un canal minier ultra large bande (UWB) en utilisant les réseaux de neurones artificiels RBF ", Québec, 2013.

[13] Y. Le Cun et al., "Back-Propagation Applied to Handwritten Zip Code Recognition, Neural Computation, Vol. 1, pp. 541-551, 1989.

Webographie

[6] NIST (1995). NIST special database 19. http: //www.nist.gov/srd/nistsd19.html.

[7] LeCun, Y. (1998). The MNIST database. http://yann.lecun.com/exdb/mnist/index.html

[12] https://www.sciencemag.org/news/2018/05/ai-researchers-allege-machine-learning-alchemy

[14] https://www.kaggle.com/