الجمهورية الشعبية الديمقراطية الجزائرية
République Algérienne démocratique et populaire

وزارة التعليم العالي و البحث العلمي
Ministère de l'enseignement supérieur et de la recherche scientifique

جـامعة سعد دحلب _ البليدة_
Université SAAD DAHLAB de BLIDA

كليـة التكنولوجيا
Faculté de Technologie

قسم الإلكترونيك
Département d'Électronique

Mémoire de Master

Filière : **Télécommunication**

Spécialité : **Réseau et Télécommunications**

Présenté par :

- LAZLI AMINE

Reconnaissance des caractères manuscrits par DEEP LEARNING

Composition du jury :

Président :Mme N. Amirouche

Encadreur : Mme I. Kaoula

Examinateur : Mme H. Bougherira

Année Universitaire 2021-2022

Remerciements

Je remercie Dieu tout puissant de m'avoir accordé volonté et patience dans l'accomplissement de ce travail.

J'exprime ma profonde gratitude à mon encadreur, Madame BOUGHERIRA, pour l'intérêt qu'elle a porté à ce travail, pour son son soutien, sa compréhension, ses orientations, et pour ses conseils au long de la réalisation de ce projet de fin d'étude. Je remercie également mes parents, mon frère ma sœur pour toute l'aide qu'ils m'ont apportée.

Mes vifs remerciements vont également à Madame Amirouche enseignante à l'université de Blida, qui m'a fait l'honneur de présider le jury de ce mémoire. Je remercie également madame Kaoula enseignante à l'université BLIDA, qui a accepté de juger mon mémoire.

Tous les mots restent faibles pour exprimer ma profonde reconnaissance à tous ceux qui m'ont aidé de près ou de loin à réaliser ce travail.

ملخص:

.العميق التعلم تقنية باستخدام اليد بخط المكتوب للنص رقمي محول تصميم هو مشروعنا من الهدف

جيدة وكانت نتائجنا تحليل على عملنا ثم .خبرة أيضًا ولدينا ، بنية بتنفيذ وقمنا ، إأم بيانات قاعدة على عملنا لهذا وواعدة

Résumé:

Le but de notre projet est de concevoir un numériseur de texte manuscrit à l'aide de la technologie de deep learning. Pour cela nous avons travaillé sur la base de donnée IAM , et nous avons implémenté une architecture Nous avons également obtenue de l'expérience.
Puis nous avons travaillé sur l'analyse de nos résultats, et ces derniers étaient bons et prometteur.

Mots clés : Deep learnnig ; Reconnaissance de texte manuscrit; Apprentissage

Abstract :

The goal of our project is to design a handwritten text digitizer using deep learning technology. For this we worked on the IAM database, and we implemented an architecture We also obtained experience. Then we worked on the analysis of our results, and these were good and promising.

Keywords: Deep learning ; Handwritten text recognition; learning

Table de matières

Liste des figures

Liste des tableaux

INTRODUCTION GENERALE

INTRODUCTION GENERALE :

L'écriture, qui a été le mode le plus naturel de collecte, de stockage et de transmission de l'information à travers les siècles et sert désormais non seulement à la communication entre les humains, mais aussi à la communication des humains et des machines.

L'effort de recherche intensif dans le domaine de la reconnaissance de caractères n'était pas seulement dû à son défi sur la simulation de la lecture humaine, mais aussi parce qu'il fournit des applications efficaces telles que le traitement automatique d'une grande quantité de papiers, le transfert de données dans des machines et l'interface Web vers le papier. La difficulté réside dans la reconnaissance des différents styles d'écriture des personnes.

L'objectif de notre travail est d'utiliser avec efficacité des techniques du deep learning pour la reconnaissance de caractères, de mots et textes manuscrits, parmi ces techniques on compte les réseaux neuronal classique, les réseaux de neurones convolutionnels, les réseaux de neurones récurrents, réseaux antagonistes génératifs ou bien l'apprentissage par renforcement profond, pour notre projet nous utiliseront les réseaux de neurones récurrents et convolutionnels.

Contrairement aux méthodes d'apprentissage automatique classique, le deep learning peut extraire automatiquement et implicitement la caractéristique d'une image, et traite directement le contenu des images.

Nous utiliserons l'un des cadres bien connus de l'apprentissage profond dans un ensemble de données réelles et discuterons des résultats.

Ce mémoire est organisé en trois chapitres :

Le premier chapitre présente l'état de l'art des méthodes de reconnaissances de caractères latins ainsi que les méthodes neuromorphiques utilisées, entre autres les méthodes du deep learning.

Dans le deuxième chapitre nous expliquons l'architecture de notre réseau de neurones profond ainsi que la base de donnée utilisée afin d'entrainer ce dernier.

Le dernier chapitre porte sur l'implémentation des algorithmes conçus pour réaliser notre programme de reconnaissance de texte manuscrit, ainsi que les résultats obtenus.

Chapitre I :

Etat de l’art

I.1 Introduction :

Le but de notre travail étant le développement d'un numériseur de textes manuscrits, nous présentons dans ce chapitre un état de l'art des méthodes de reconnaissances de caractères latins, de mots, et leur classification. Nous donnons ensuite les méthodes neuromorphiques utilisés, et plus particulièrement les méthodes de deep learning.

I.2 Histoire :

La RTM (reconnaissance de texte manuscrit) est un domaine de recherche actif dans les sciences informatiques, remontant au milieu du vingtième siècle.

La RTM était à l'origine extrêmement lié au développement de la technologie de reconnaissance optique de caractères (ROC), où les images numérisées du texte imprimé sont converti en texte codé par machine, généralement en comparant des caractères individuels avec modèles existants [1].

La RTM est devenu un domaine de recherche à part entière en raison de la variabilité des différentes mains, et la complexité informatique de la tâche [1]. Les progrès statistiques dans les années 1980 et les avancées la reconnaissance de formes combinée à l'intelligence artificielle dans les années 1990 ont été suivies par le développement des approches de réseaux de neurones profonds dans les années 2000 et 2010. Ceci, combiné avec la disponibilité d'une puissance de traitement informatique accrue, a entraîné des améliorations dans la reconnaissance des documents historiques manuscrits, comme en témoignent régulièrement les concours scientifiques des deux principales conférences dans ce domaine :

La conférence internationale sur l'analyse et la reconnaissance de documents et la Conférence internationale sur Frontières de la reconnaissance de l'écriture manuscrite. Les chercheurs ont initialement développé cette technologie avec des matériaux manuscrits à l'esprit et il est largement connu dans le domaine de l'informatique sous les initiales HTR.[1]

I.3 Applications de la reconnaissance de l'écriture manuscrite :

- **Secteur bancaire :**

Le secteur bancaire est un grand consommateur de reconnaissance d'écriture manuscrite (RM). L'utilisation la plus fréquente de la reconnaissance de l'écriture manuscrite est le traitement des chèques : un chèque manuscrit est scanné, son contenu converti en texte numérique, la signature vérifiée et le chèque nettoyé en temps réel, le tout avec une implication humaine. Alors qu'une précision de près de 100 % a été atteinte pour les chèques imprimés [2].

- **Santé:**

Le domaine de la santé est un domaine qui se débrouille bien avec la reconnaissance de l'écriture manuscrite. Avoir tous ses antécédents médicaux sur un magasin numérique consultable signifie que des éléments tels que les maladies et les traitements antérieurs, les tests de diagnostic, les dossiers hospitaliers, les paiements d'assurance, etc. peuvent être mis à disposition dans un seul endroit, plutôt que d'avoir à conserver des fichiers de rapports encombrants, X- rayons et autres papiers [2].

- **Documents administratifs** :

Peu d'industries génèrent autant de paperasse que l'industrie juridique, et donc l'écriture manuscrite la reconnaissance a ici de multiples applications. Des tonnes et des tonnes d'affidavits, de jugements, les dépôts, les déclarations, les testaments et autres documents juridiques, en particulier les imprimés, peuvent être numérisé, stocké, mis en base de données et rendu consultable à l'aide du plus simple des lecteurs de reconnaissance d'écriture manuscrite [2].

I.4 Reconnaissance en ligne et hors ligne :

En général, la reconnaissance de l'écriture manuscrite est classée en deux types de reconnaissance de caractères hors ligne et en ligne. Hors ligne la reconnaissance de l'écriture manuscrite implique la conversion automatique de texte dans une image en codes de lettre qui sont utilisables dans applications informatiques et de traitement de texte. La reconnaissance de caractères en ligne traite un flux de données qui provient d'un transducteur pendant que l'utilisateur écrit. Le matériel typique pour collecter des données est une tablette numérique qui est électromagnétique ou sensible à la pression. Lorsque l'utilisateur écrit sur la tablette, les mouvements successifs du stylet sont transformés en une série de signaux électroniques qui est mémorisé et analysé par l'ordinateur [3].

I.4.1 Système de reconnaissance de caractères en ligne :

La reconnaissance de caractères en ligne fait référence au processus de reconnaître l'écriture manuscrite enregistrée avec un numériseur comme une heure séquence de coordonnées de stylo.
L'écriture manuscrite est capturée et stockée sous forme numérique via différents moyens. Habituellement, un stylo spécial est utilisé en conjonction avec une surface électronique. Comme le stylo se déplace sur la surface, les coordonnées bidimensionnelles de points successifs sont représentées en fonction du temps et sont stockés dans l'ordre Il est généralement admis que le procédé en ligne de reconnaissance de texte manuscrit a atteint de meilleurs résultats que son homologue hors ligne. Cela

peut être attribué au fait que plus d'informations peuvent être capturées dans le cas en ligne tels que la direction, la vitesse et l'ordre de traits de l'écriture manuscrite [3].

I. 4.2 Système de reconnaissance de caractères hors ligne :

La reconnaissance de l'écriture manuscrite hors ligne fait référence au processus de reconnaître des mots qui ont été scannés à partir d'une surface (comme une feuille de papier) et sont stockés numériquement en gris format d'échelle. Après avoir été stocké, il est classique d'effectuer d'autres traitement pour permettre une meilleure reconnaissance. La reconnaissance de caractères hors ligne peut être regroupée en deux types[3] :

- Reconnaissance magnétique des caractères (MCR)
- Reconnaissance optique de caractères (OCR)

Dans MCR, les caractères sont imprimés avec de l'encre magnétique. Le dispositif de lecture peut reconnaître les caractères selon le champ magnétique unique de chaque personnage. Le MCR est principalement utilisé dans les banques pour l'authentification des chèques. L'OCR s'occupe de la reconnaissance de caractères acquis par voie optique, généralement un scanner ou un appareil photo. Les personnages sont dans sous forme d'images pixélisées, et peuvent être imprimées ou manuscrite, dans n'importe quelle taille, forme ou orientation.[3]

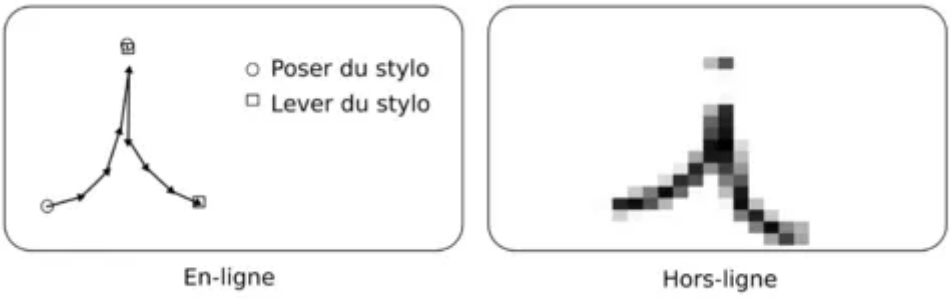


Figure 1 : Différence entre Reconnaissance en ligne et hors-ligne [4]

I.5 Différences entre les systèmes de reconnaissance en ligne et hors ligne :

Nous incluons les différences entre la reconnaissance en ligne et la reconnaissance hors ligne dans le tableau 1 ci-dessous [3] :

Tableau 1 : Différences entre la reconnaissance en ligne et la reconnaissance hors ligne

	Comparaison	Caractères en ligne	Caractères hors ligne
1-	Disponibilité du nombre de traits de stylet	Oui	Non
2-	Façon d'écrire	En utilisant stylo numérique sur LCD	Papier document
3-	Taux de reconnaissance	Plus élevé	Moindre
4-	Précision	Plus élevé	Moindre

I.6 Contraintes spécifiques de l'écriture manuscrite :

L'identification et l'extraction d'informations manuscrites restent un défi dans le processus de numérisation. Les raisons en sont nombreuses : les documents originaux peuvent être de mauvaise qualité car le papier se détériore facilement ; les notes peuvent avoir été écrites de façon bâclée ; les signatures sont presque toujours illisibles.

Quels que soient les efforts et le travail que nous déployons pour créer un algorithme de reconnaissance de texte manuscrit efficace, il ne sera jamais parfait et sans marge d'erreur, principalement en raison des types différents et étonnants de styles et de polices d'écriture manuscrite que des millions de personnes autour du monde avec chaque individu ayant son style d'écriture unique (figure 2 et 3), donc faire l'algorithme parfait exigerait une immense base de données à partir de laquelle notre réseau de neurones apprendrait et ce ne serait toujours pas parfait, car la taille, la police de l'écriture manuscrite ainsi que l'espacement entre les mots rendraient un peu plus difficile pour l'algorithme de reconnaître correctement les mots avec facilité et sans problème.

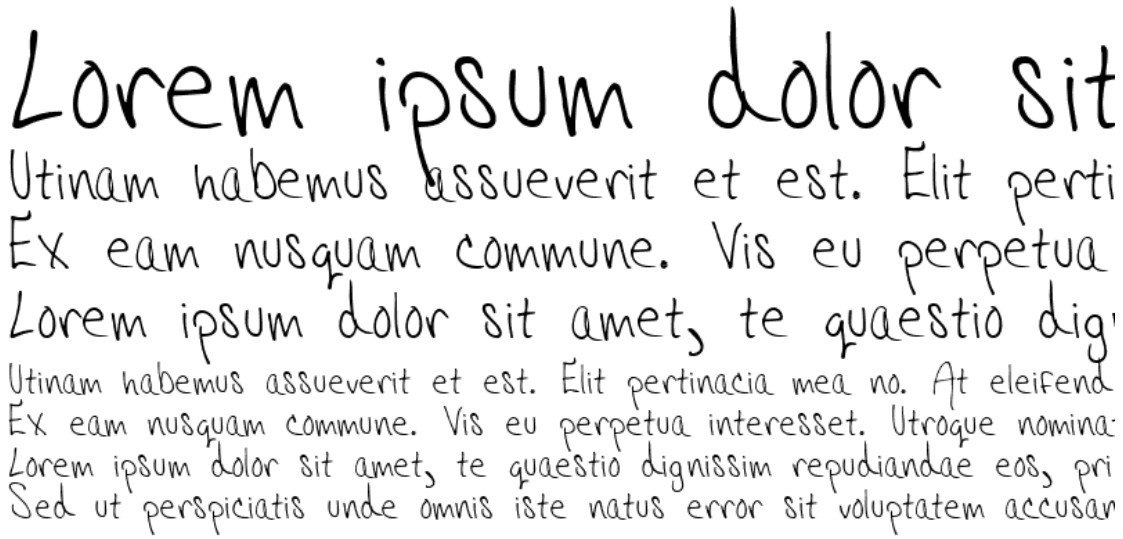


Figure 2 : Exemple d'écriture humaine aléatoire [5]

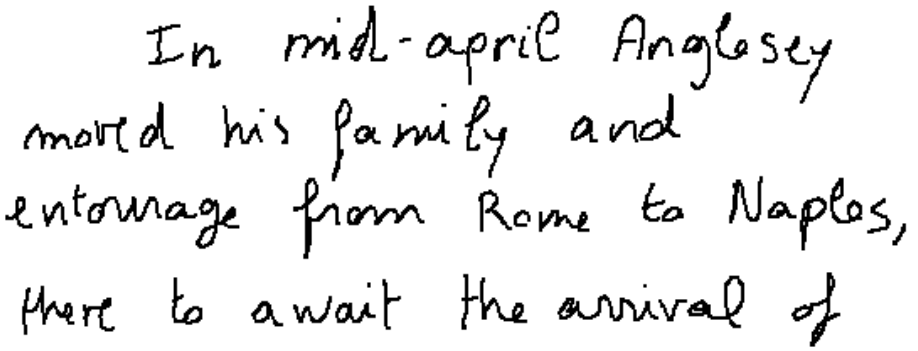


Figure 3 : Image contenante un texte manuscrit [6]

I. 7 Approches de reconnaissance :

Durant les dernières décennies, beaucoup de méthodes de segmentation ont été développées dans le but d'avoir un système de reconnaissance de caractères plus robuste. Malgré tous les efforts, la situation reste loin d'atteindre les ambitions. En se basant sur le processus de segmentation, deux approches ont été appliquées [7] :

• Approche globale.

• Approche analytique

I.7.1 L'approche globale :

L'approche globale essaye de reconnaître la représentation intégrale des mots de l'image d'entée et de le décrire indépendamment des caractères qui le constituent. Cette approche présente l'avantage de garder le caractère dans son contexte avoisinant, ce qui permet une modélisation plus efficace des variations de l'écriture et des dégradations qu'elle peut subir. Cependant cette méthode est pénalisante

par la taille de la mémoire, le temps de calcul et la complexité du traitement qui croit linéairement avec l a taille du lexique considéré, d'où une limitation du vocabulaire . [7]

I.7.2 L'approche analytique ou locale :

Contrairement à l'approche globale, l'approche analytique cherche à identifier les caractères ou sous-caractères (graphèmes) issus de la segmentation (séparation de mots, des caractères) pour reconstituer les mots.

La difficulté d'une telle approche a été clairement évoquée par Sayre en 1973 et peut être résumée par le dilemme suivant : "pour reconnaître les lettres, il faut segmenter le tracéet pour segmenter le tracé, il faut reconnaître les lettres". Cette approche est la seule applicable dans le cas de grands vocabulaires. Elle peut s'adapter facilement à un changement de vocabulaire. Elle permet théoriquement une discrimination plus fine des mots car elle se base sur la reconnaissance des lettres qui la composent et il est possible de récupérer l'orthographe du

mot reconnu. Son inconvénient principal demeure la nécessité de l'étape de segmentation avec les problèmes de sous- ou de sur-segmentation que cela implique.

Certaines des approches actuelles se proposent de tirer avantage des deux méthodes, réduisant la complexité de l'approche globale en l'appliquant sur des entités plus petites (lettres). L'approche analytique recherche la séquence de lettres contenues dans l'image à reconnaître. Certains modèles permettent de combiner ces deux niveaux en un seul et peuvent ainsi s'affranchir de la segmentation préalable de l'image. [7]

I.7.3 La classification :

Reconnaître un objet est lui assigner une étiquette, un label, une classe. La reconnaissance de l'écriture est donc un problème de classification, pour une image d'un mot ou d'une phrase nous désirons déterminer l'ensemble des caractères contenus dans cette image.

La classification est l'opération de base d'un système de reconnaissance de caractères. Son principe consiste à partir d'une source d'entrée constituée de plusieurs caractères différents dans notre cas à reconnaitre chacun d'entre eux et à lui attribuer, en sortie, une classe propre à chaque caractère. Deux caractères différents (exemple : Lettre A et Lettre B) seront donc rangés dans deux classes différentes. Pour pouvoir être distingués l'un de l'autre par le classifieur, chaque lettre doit être caractérisée par des paramètres pertinents choisis de telle sorte qu'il n'y ait pas de risque de confusion entre elles. Une fois ces paramètres définis, le classifieur doit, dans un premier temps, Apprendre à les reconnaitre. Cette

étape dite, phase d'apprentissage, consiste à injectés à l'entrée du système, l'ensemble des caractères de la source autant de fois qu'il le faut jusqu'à ce que la structure interne du classifieur, qui s'auto-

corrige au fur et à mesure à l'étape de reconnaissance. Durant cette dernière, chaque caractère de la source d'entrée pourra alors être reconnu et rangé dans la classe qui lui correspond.

Le rôle d'un classifieur est de déterminer, parmi un ensemble fini de classe, à laquelle appartient un objet donné. Un tel système doit être capable de modéliser les frontières qui séparent les classes les unes des autres. Cette modélisation est appelée fonction discriminante, à partir de laquelle la classification s'effectue

De nombreux classifieurs existent et sont plus ou moins bien adaptés à la reconnaissance de l'écriture, on peut citer les SVM (Machines à Vaste marge), les classifieurs statistiques par HMM (Hidden MarkovModel) et les RDN (Réseaux de Neurones) que nous décrirons de façon plus détaillée puisque nous utiliseront les réseaux de neurones profond dans notre projet. [8]

I.7.4 Les SVMs

Les machines à vecteur de support appelées aussi classificateur à marge optimale ou encore séparateurs à vaste marge, SVM ont été introduites par Vapnik. Leur principe est de maximiser la marge entre les classes. Il faut donc déterminer l'hyperplan maximisant cette marge. Les SVM offrent des performances intéressantes pour la reconnaissance de caractère manuscrites, mais ils sont peu applicables à la reconnaissance de mots (sauf éventuellement avec une segmentation explicite des mots en lettres). En effet, ils travaillent avec des données en dimension fixe et ne permettent donc pas d'introduire la variabilité de longueur des mots. De plus ils ont l'inconvénient d'avoir un temps de calcul assez long en phase d'apprentissage comme en phase de reconnaissance. [8]

I.7.5 Les approches markoviennes

Les modèles Markov sont couramment utilisés pour la reconnaissance de l'écriture manuscrite. En effet, ce sont des outils statistiques puissants permettant de calculer la probabilité d'appartenance d'une forme à une classe. La forme est vue comme un ensemble d'observation émises par des états cachés. La modélisation markovienne est bien adaptée à l'écriture manuscrite car elle permet d'intégrer les variabilités de longueur des mots . [8]

I.8 Les réseaux de neurones :

I.8.1 Le neurone artificiel :

Un neurone artificiel est un point de connexion dans un réseau neuronal artificiel. Les réseaux de neurones artificiels, comme le réseau neuronal biologique du corps humain, ont une architecture en couches et chaque nœud de réseau (point de connexion) a la capacité de traiter l'entrée et de transmettre la sortie à d'autres nœuds du réseau. Dans les architectures artificielles et biologiques, les nœuds sont appelés neurones et les connexions sont caractérisées par des poids synaptiques, qui représentent l'importance de la connexion. Au fur et à mesure que de nouvelles données sont reçues et traitées, les poids synaptiques changent et c'est ainsi que l'apprentissage se produit. [9]

La figure 4 illustre l'architecture d'un réseau de neurones artificiel.

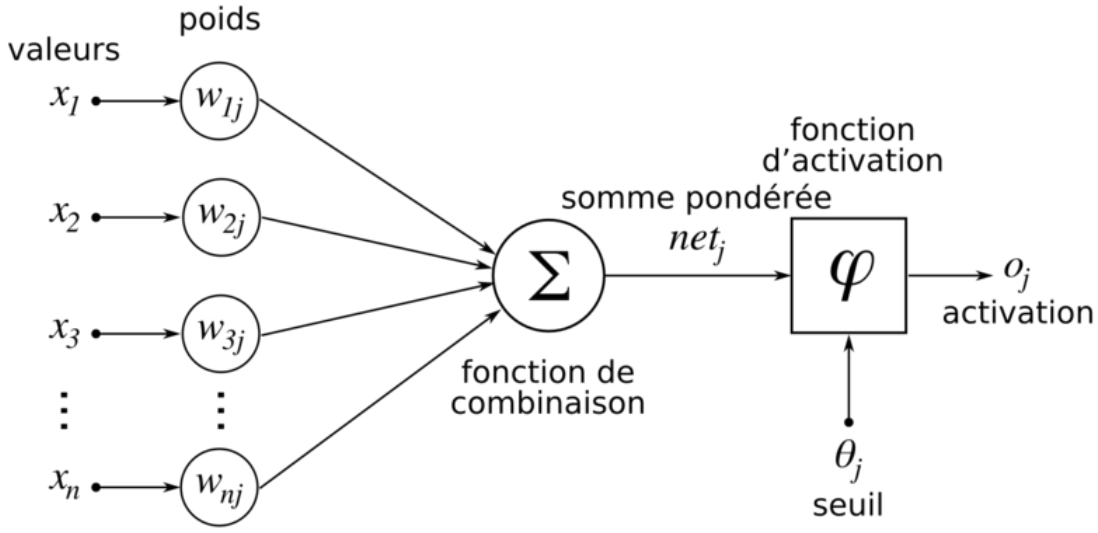


Figure 4: Architecture d'un neurone [10]

I.8.2 Perceptron multi-couches :

Les réseaux de perceptrons multicouches appartiennent à la classe des classificateurs neuronaux supervisés. Ils sont constitués de perceptrons organisés en couches : une couche d'entrée, une ou plus de couches cachées et la couche de sortie. Chaque perceptron dans une couche particulière est généralement connecté à chaque perceptron dans la couche supérieure et inférieure. Ces les connexions portent des poids 'wi'. Chaque perceptron calcule la somme des entrées, et l'alimente dans sa fonction d'activation, régulièrement sigmoïde.

Le résultat est ensuite transmis à la couche suivante. La couche de sortie a, par exemple, le même nombre de perceptrons comme il y a des classes, et le perceptron avec le plus haut l'activation tiendra compte de la classification de l'échantillon d'entrée. [11]

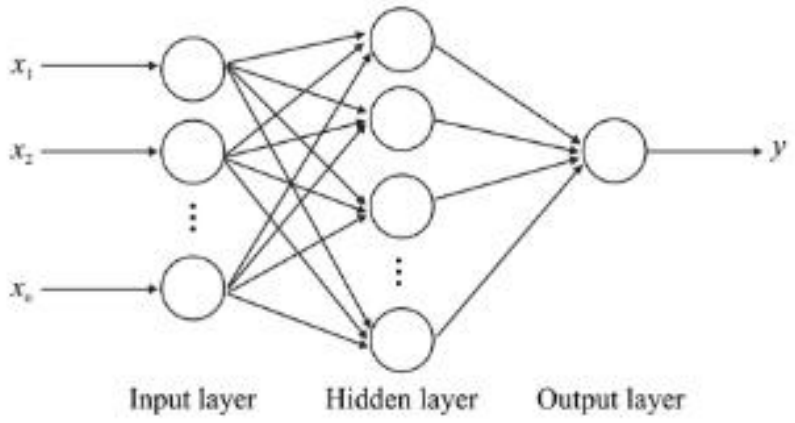


Figure 5 : Architecture d'un perceptron multicouche [12]

I.8.3 Les fonctions d'activations des réseaux de neurones :

Une fonction d'activation dans un réseau de neurones définit comment la somme pondérée de l'entrée est transformée en une sortie d'un nœud ou de nœuds dans une couche du réseau.

Le choix de la fonction d'activation a un impact important sur la capacité et les performances du réseau neuronal, et différentes fonctions d'activation peuvent être utilisées dans différentes parties du modèle.

Techniquement, la fonction d'activation est utilisée dans ou après le traitement interne de chaque nœud du réseau, bien que les réseaux soient conçus pour utiliser la même fonction d'activation pour tous les nœuds d'une couche. [13]

I.8.4 Types de fonctions d'activations :

Plusieurs fonctions d'activation qui peuvent être utilisées avec les réseaux de neurones sont montrées ci-dessous (figure 6) :

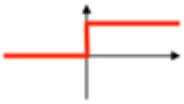
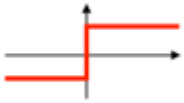
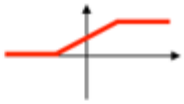
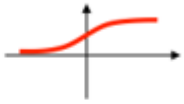
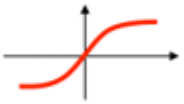
Activation function	Equation	Example	1D Graph
Unit step (Heaviside)	$\phi(z) = \begin{cases} 0, & z < 0, \\ 0.5, & z = 0, \\ 1, & z > 0, \end{cases}$	Perceptron variant	
Sign (Signum)	$\phi(z) = \begin{cases} -1, & z < 0, \\ 0, & z = 0, \\ 1, & z > 0, \end{cases}$	Perceptron variant	
Linear	$\phi(z) = z$	Adaline, linear regression	
Piece-wise linear	$\phi(z) = \begin{cases} 1, & z \geq \frac{1}{2}, \\ z + \frac{1}{2}, & -\frac{1}{2} < z < \frac{1}{2}, \\ 0, & z \leq -\frac{1}{2}, \end{cases}$	Support vector machine	
Logistic (sigmoid)	$\phi(z) = \frac{1}{1 + e^{-z}}$	Logistic regression, Multi-layer NN	
Hyperbolic tangent	$\phi(z) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}}$	Multi-layer Neural Networks	
Rectifier, ReLU (Rectified Linear Unit)	$\phi(z) = max(0, z)$	Multi-layer Neural Networks	

Figure 6 : Quelques fonctions d'activation régulièrement utilisées dans la construction réseaux de neurones[14]

I.9 Méthodes d'apprentissage :

L'apprentissage est une phase du développement d'un réseau de neurones durant laquelle le comportement du réseau est modifié jusqu'à l'obtention du comportement désiré. On distingue parmi les types d'apprentissage (figure 7):

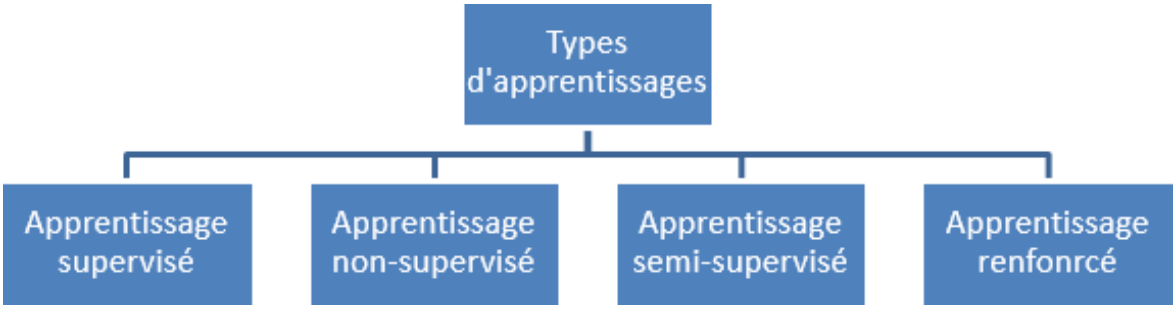


Figure 7 : Schéma synoptique des types d'apprentissages [15]

- **L’apprentissage automatique (machine learning):**

Le Machine Learning ou apprentissage automatique est un domaine scientifique, et plus particulièrement une sous-catégorie de l’intelligence artificielle. Elle consiste à laisser des algorithmes découvrir des ” patterns “, à savoir des motifs récurrents, dans les ensembles de données. Ces données peuvent être des chiffres, des mots, des images, des statistiques.

Tout ce qui peut être stocké numériquement peut servir de données pour le Machine Learning. En décelant les patterns dans ces données, les algorithmes apprennent et améliorent leurs performances dans l’exécution d’une tâche spécifique. [16]

Exemples :

-La reconnaissance des caractères manuscrite

- Prédire un comportement face à une nouvelle donnée

- **Apprentissage Supervisé :**

Les techniques d’apprentissage supervisé permettent de construire des modèles à partir d’exemples d’apprentissage ou d’entrainement dont on connait le comportement ou la réponse. Ces modèles peuvent, ensuite, être utilisés dans différentes applications, telles que la prédiction ou la classification.[17]

- **Apprentissage Non-Supervisé :**

Il vise à concevoir un modèle structurant l’information. La différence est que les comportements (ou catégories ou encore les classes) des données d’apprentissage ne sont pas connus, c’est ce que l’on cherche à trouver.[17]

- **L’apprentissage semi supervisé :**

Il prend en entrée certaines données annotées et d'autres non. Ce sont des méthodes très intéressantes qui tirent parti des deux mondes (supervisé et non-supervisé), mais apportent leur lot de difficultés.[15]

- **L’apprentissage renforcé :**

Il se base sur un cycle d'expérience / récompense et améliore les performances à chaque itération. Une analogie souvent citée est celle du cycle de dopamine : une "bonne" expérience augmente la dopamine et donc augmente la probabilité que l'agent répète l'expérience.[15]

I.10 Domaines d'applications des réseaux de neurones :

Les réseaux de neurones s'avèrent plus performants que les techniques de régressions pour les taches de machine learning.[18]

Les domaines d'application des réseaux neuronaux sont souvent caractérisés par une relation entrée-sortie de la donnée d'information :

- La reconnaissance d'image
- Les classifications de textes ou d'images
- Identification d'objets
- Prédiction de données
- Filtrage d'un set de données

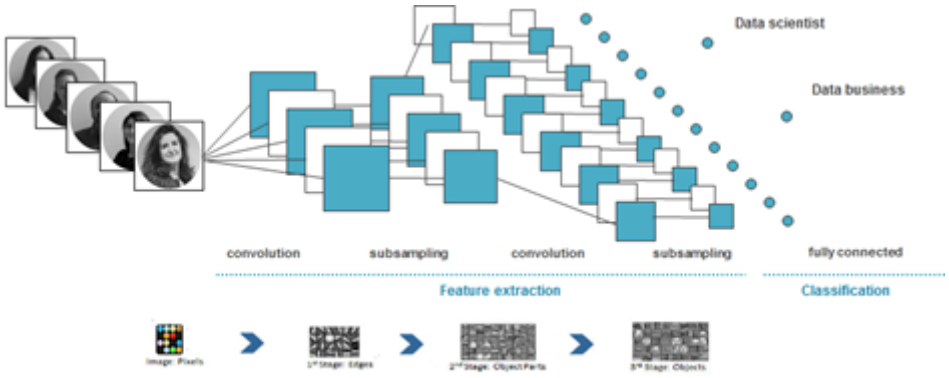


Figure 8 : Réseaux de neurones classiques pour reconnaissance d'image [18]

I.11 Le deep learning :

Le Deep Learning est un type d'apprentissage machine (ML) et d'intelligence artificielle (IA) qui imite la façon dont les humains acquièrent certains types de connaissances. Le Deep Learning est un élément important de la science des données, qui comprend les statistiques et la modélisation prédictive. Il est extrêmement utile aux scientifiques chargés de collecter, d'analyser et d'interpréter de grandes quantités de données ; Le Deep Learning rend ce processus plus rapide et plus facile.

Dans sa forme la plus simple, Le Deep Learning peut être considéré comme un moyen d'automatiser l'analyse prédictive. Alors que les algorithmes traditionnels d'apprentissage machine sont linéaires, les algorithmes de Deep Learning sont empilés dans une hiérarchie de complexité et d'abstraction croissantes.

Pour comprendre Le Deep Learning, nous imaginons un enfant dont le premier mot est chien. Le jeune enfant apprend ce qu'est un chien – et ce qu'il n'est pas – en pointant des objets et en prononçant le mot chien. Le parent dit : « Oui, c'est un chien » ou « Non, ce n'est pas un chien ». En continuant à montrer des objets du doigt, le jeune enfant devient plus conscient des caractéristiques que possèdent tous les chiens. Ce que le jeune enfant fait, sans le savoir, c'est clarifier une abstraction complexe – le concept de chien – en construisant une hiérarchie dans laquelle chaque niveau d'abstraction est créé avec les connaissances acquises au niveau précédent de la hiérarchie.[19]

I.11.1 Fonctionnement du deep learning :

Les programmes informatiques qui utilisent Le Deep Learning passent par le même processus que l'apprentissage du jeune enfant pour identifier le chien. Chaque algorithme de la hiérarchie applique une transformation non linéaire à son entrée et utilise ce qu'il apprend pour créer un modèle statistique en sortie. Les itérations se poursuivent jusqu'à ce que la sortie ait atteint un niveau de précision acceptable. Le nombre de couches de traitement par lesquelles les données doivent passer (figure 9) est ce qui a inspiré le mot « deep ».

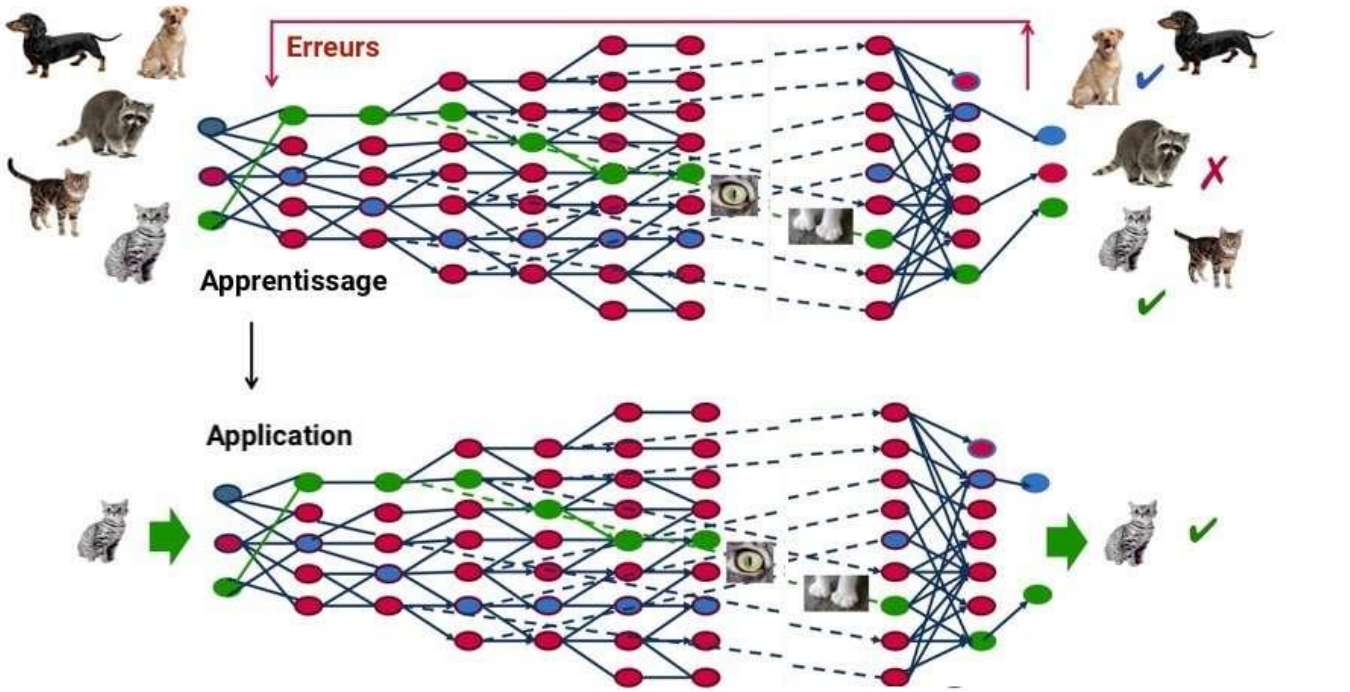


Figure 9 : Identification d'un chat a partir d'un processus d'aprentissage .[19]

Dans l'apprentissage machine traditionnel, le processus d'apprentissage est supervisé, et le programmeur doit être extrêmement précis lorsqu'il indique à l'ordinateur le type de choses qu'il doit rechercher pour décider si une image contient un chien ou non. Il s'agit d'un processus laborieux appelé extraction de caractéristiques, et le taux de réussite de l'ordinateur dépend entièrement de la capacité du programmeur

à définir avec précision un ensemble de caractéristiques pour « chien ». L'avantage de Le Deep Learning est que le programme construit lui-même l'ensemble de caractéristiques sans supervision. L'apprentissage non supervisé est non seulement plus rapide, mais il est aussi généralement plus précis.

Contrairement au jeune enfant, qui mettra des semaines, voire des mois, à comprendre le concept de « chien », un programme informatique qui utilise des algorithmes d'apprentissage profond peut se voir montrer un ensemble de formation et trier des millions d'images, identifiant avec précision en quelques minutes les images qui contiennent des chiens.

Pour atteindre un niveau de précision acceptable, les programmes de Deep Learning nécessitent l'accès à d'immenses quantités de données d'entraînement et de puissance de traitement, qui n'étaient pas facilement accessibles aux programmeurs avant l'ère du big data et de l'informatique dématérialisée. Comme les programmes de deep learning peuvent créer des modèles statistiques complexes directement à partir de leurs propres résultats itératifs, ils sont capables de créer des modèles prédictifs précis à partir de grandes quantités de données non étiquetées et non structurées. C'est important car l'internet des objets (IoT) continue à devenir plus omniprésent, car la plupart des données que les humains et les machines créent sont non structurées et ne sont pas étiquetées. .[19]

I.11.2 Réseaux de neurones convolutionnels (CNN) :

Dans le Deep Learning, un réseau de neurones convolutifs ou CNN est un type de réseau de neurones artificiels largement utilisé pour la reconnaissance et la classification d'images/d'objets. Le Deep Learning reconnaît ainsi des objets dans une image en utilisant un CNN.

Étant donné que les CNN sont les modèles largement utilisés pour effectuer les tâches de traitement d'image et de vision par ordinateur, nous avons également de bons modèles pour les tâches de détection d'objets basées sur CNN. Dans le cas de la détection d'objets où CNN est très utile, nous avons trois modèles CNN basés sur la région que nous appelons R-CNN, Fast R-CNN et Faster R-CNN. [20]

I.11.2.1 Méthodes de détection :

I.11.2.2 R-CNN

Les R-CNN (Region-based Convolutional Neural Networks) sont une famille de modèles d'apprentissage automatique utilisés dans la vision par ordinateur et le traitement d'images. Spécialement conçu pour la détection d'objets, l'objectif initial de tout R-CNN est de détecter des objets dans n'importe quelle image d'entrée définissant les limites qui les entourent, c'est aussi le modèle qu'on utilisera plus tard dans notre expérience.[20]

I.11.2.3 Fast R-CNN :

Comme dans la détection R-CNN, la détection Fast R-CNN utilise également un algorithme comme Edge Boxes pour générer des propositions de régions. Contrairement au détecteur R-CNN, qui recadre et redimensionne les propositions de régions, le détecteur Fast R-CNN traite l'intégralité de l'image. Alors qu'un détecteur R-CNN doit classer chaque région, Fast R-CNN regroupe les

fonctionnalités CNN correspondant à chaque proposition de région. Fast R-CNN est plus efficace que R-CNN, car dans le détecteur Fast R-CNN, les calculs pour les régions qui se chevauchent sont partagés[20].

I.11.2.4 Faster R-CNN :

Le Faster R-CNN est la meilleure méthode de détection d'objets parmi tous les algorithmes améliorés basés sur le R-CNN. Ce dernier recherche des zones candidates en introduisant un réseau de proposition de région au lieu d'une recherche sélective. Les cartes de caractéristiques obtenues à partir des images d'entrée sont passées à travers le réseau neuronal convolutif, puis ces cartes sont envoyées au réseau de suggestion régional. [21]

I.11.3 Exemples d'applications de Deep Learning :

Comme les modèles d'apprentissage profond traitent l'information de manière similaire au cerveau humain, ils peuvent être appliqués à de nombreuses tâches que les gens accomplissent. Le Deep Learning est actuellement utilisé dans la plupart des outils de reconnaissance d'images, de traitement du langage naturel et des logiciels de reconnaissance vocale. Ces outils commencent à apparaître dans des applications aussi diverses que les voitures et les services de traduction de langues.[19]

I.11.4 Domaines d'applications du deep learning :

Les cas d'utilisation actuels du Deep Learning comprennent tous les types de grandes applications d'analyse de données, en particulier celles axées sur le traitement du langage naturel, la traduction des langues, le diagnostic médical, les signaux boursiers, la sécurité des réseaux et la reconnaissance d'images. Parmi les domaines spécifiques dans lesquels le Deep Learning est actuellement utilisé, on peut citer les suivants :

- **Expérience client** : Des modèles de Deep Learning sont déjà utilisés pour les chatbots. Et, au fur et à mesure qu'il se développe, Le Deep Learning devrait être mis en œuvre dans diverses entreprises pour améliorer l'expérience des clients et accroître leur satisfaction.
- **Génération de texte** : Les machines apprennent la grammaire et le style d'un texte et utilisent ensuite ce modèle pour créer automatiquement un texte entièrement nouveau correspondant à l'orthographe, à la grammaire et au style du texte original.
- **Aéronautique et militaire** : Le Deep Learning est utilisé pour détecter des objets provenant de satellites qui identifient des zones d'intérêt, ainsi que des zones sûres ou non pour les troupes.
- **Automatisation industrielle** : Le Deep Learning améliore la sécurité des travailleurs dans des environnements tels que les usines et les entrepôts en fournissant des services qui détectent automatiquement lorsqu'un travailleur ou un objet s'approche trop près d'une machine.

- **Ajouter de la couleur** : Il est possible d'ajouter de la couleur aux photos et vidéos en noir et blanc à l'aide de modèles de Deep Learning. Dans le passé, il s'agissait d'un processus manuel extrêmement long.
- **La recherche médicale** : Les chercheurs en cancérologie ont commencé à mettre en œuvre Le Deep Learning dans leur pratique afin de détecter automatiquement les cellules cancéreuses.
- **La vision par ordinateur** : Le Deep Learning a considérablement amélioré la vision par ordinateur, en fournissant aux ordinateurs une précision extrême pour la détection d'objets et la classification, la restauration et la segmentation d'images.[19]

I.11.5 Architectures des réseaux de neurones profonds :

Les réseaux de neurones multicouches ou profonds peuvent comporter des millions de neurones, répartis en plusieurs dizaines de couches. Ils sont utilisés en deep learning pour concevoir des mécanismes d'apprentissage supervisés et non supervisés.

Dans ces architectures mathématiques, chaque neurone effectue des calculs simples mais les données d'entrées passent à travers plusieurs couches de calcul avant de produire une sortie. Les résultats de la première couche de neurones servent d'entrée au calcul de la couche suivante et ainsi de suite. Il est possible de jouer sur les différents paramètres de l'architecture du réseau : le nombre de couches, le type de chaque couche, le nombre de neurones qui composent chaque couche.

Un réseau de neurones profond est une succession de couches de convolution, pooling, correction, réseau récurrent, réseau entièrement connecté [22]

I.11.6 Architecture de réseaux de neurones convolutionnel :

Les réseaux de neurones convolutionnels sont basés sur le perceptron multicouche (MLP), et inspirés du comportement du cortex visuel des vertébrés. Bien qu'efficaces pour le traitement d'images, les MLP ont beaucoup de mal à gérer des images de grande taille, ce qui est dû à la croissance exponentielle du nombre de connexions avec la taille de l'image

. Par exemple, si on prend une image de taille 32x32x3 (32 de large, 32 de haut, 3 canaux de couleur), un seul neurone entièrement connecté dans la première couche cachée du MLP aurait 3072 entrées (32*32*3). Une image 200x200 conduirait ainsi à traiter 120 000 entrées par neurone ce qui, multiplié par le nombre de neurones, devient énorme.

Les CNN visent à limiter le nombre d'entrées tout en conservant la forte corrélation « spatialement locale » des images naturelles. Par opposition aux MLP, les CNN ont les traits distinctifs suivants[23] :

1. 'Volumes 3D de neurones’. La couche de neurones n'est plus simplement une surface (perceptron), mais devient un volume avec une profondeur. Si on considère un seul champ récepteur du CNN, les n neurones associés (sur la profondeur) forment l'équivalent de la première couche d'un MLP.

2. 'Connectivité locale'. Grâce au champ récepteur qui limite le nombre d'entrées du neurone, tout en conservant l'architecture MLP, les CNN assurent ainsi que les « filtres» produisent la réponse la plus forte à un motif d'entrée spatialement localisé, ce qui conduit à une représentation parcimonieuse de l'entrée. Une telle représentation occupe moins d'espace en mémoire. De plus, le nombre de paramètres à estimer étant réduit, leur estimation (statistique) est plus robuste pour un volume de données fixé (comparé à un MLP).

3. 'Poids partagés' : Dans les CNN, les paramètres de filtrage d'un neurone (pour un champ récepteur donné) sont identiques pour tous les autres neurones d'un même noyau (traitant tous les autres champs récepteurs de l'image). Ce paramétrage (vecteur de poids et biais) est défini dans une « carte de fonction ». Cela signifie que tous les neurones dans une couche de convolution donnée détectent exactement la même caractéristique. En multipliant les champs récepteurs, il devient possible de détecter des éléments indépendamment de leur position dans le champ visuel, ce qui induit une propriété d'invariance par translation.[23]

Ensemble, ces propriétés permettent aux réseaux de neurones convolutionnels d'obtenir une meilleure généralisation (en termes d'apprentissage) sur des problèmes de vision. Le partage de poids permet aussi de réduire considérablement le nombre de paramètres libres à apprendre, et ainsi les besoins en mémoire pour le fonctionnement du réseau. La diminution de l'empreinte mémoire permet l'apprentissage de réseaux plus grands donc souvent plus puissants.

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 la taille de l'image intermédiaire (souvent par sous-échantillonnage).

- 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).[23]

I.12 Les types de couches d'un réseau CNN :

I.12.1Couche de convolution :

La couche de convolution est le bloc de construction de base d'un CNN. Trois paramètres permettent de dimensionner le volume de la couche de convolution la profondeur, le pas et la marge :

1. Profondeur de la couche : nombre de noyaux de convolution (ou nombre de neurones associés à un même champ récepteur).

2. Le pas: contrôle le chevauchement des champs récepteurs. Plus le pas est petit, plus les champs récepteurs se chevauchent et plus le volume de sortie sera grand.

3. La marge (à 0) ou zero padding : parfois, il est commode de mettre des zéros à la frontière du volume d'entrée. La taille de ce 'zero-padding' est le troisième hyper paramètre. Cette marge permet de contrôler la dimension spatiale du volume de sortie. En particulier, il est parfois souhaitable de conserver la même surface que celle du volume d'entrée.

Si le pas et la marge appliquée à l'image d'entrée permettent de contrôler le nombre de champs récepteurs à gérer (surface de traitement), la profondeur permet d'avoir une notion de volume de sortie, et de la même manière qu'une image peut avoir un volume, si on prend une profondeur de 3 pour les trois canaux RGB d'une image couleur, la couche de convolution va également présenter en sortie une profondeur. C'est pour cela que l'on parle plutôt de "volume de sortie" et de "volume d'entrée", car l'entrée d'une couche de convolution peut être soit une image soit la sortie d'une autre couche de convolution.

La taille spatiale du volume de sortie peut être calculée en fonction de la taille du volume d'entrée Wi la surface de traitement K (nombre de champs récepteurs), le pas S avec lequel ils sont appliqués, et la taille de la marge P La formule pour calculer le nombre de neurones du volume de sortie est W0 = Si W0 n'est pas entier, les neurones périphériques n'auront pas autant d'entrée que les autres. Il faudra donc augmenter la taille de la marge (pour recréer des entrées virtuelles).

Souvent, on considère un pas S=1, on calcule donc la marge de la manière suivante : P= 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".[23]

II.12.2 Couche de pooling (POOL) :

Un autre concept important des CNNs est le pooling, ce qui est une forme de sous-échantillonnage de l'image. L'image d'entrée est découpée en une série de rectangles de n pixels de côté ne se chevauchant pas (pooling). Chaque rectangle peut être vu comme une tuile. Le signal en sortie de tuile est défini en fonction des valeurs prises par les différents pixels de la tuile.

Le pooling réduit la taille spatiale d'une image intermédiaire, réduisant ainsi la quantité de paramètres et de calcul dans le réseau. Il est donc fréquent d'insérer périodiquement une couche de pooling entre deux couches convolutives successives d'une architecture CNN pour contrôler l'overfitting (sur-apprentissage). L'opération de pooling créait aussi une forme d'invariance par translation.

La couche de pooling fonctionne indépendamment sur chaque tranche de profondeur de l'entrée et la redimensionne uniquement au niveau de la surface. La forme la plus courante est une couche de mise en commun avec des tuiles de taille 2x2 (largeur/hauteur) et comme valeur de sortie la valeur maximale en entrée. On parle dans ce cas de « Max-Pool 2x2 »

Il est possible d'utiliser d'autres fonctions de pooling que le maximum. On peut utiliser un « average pooling » (la sortie est la moyenne des valeurs du patch d'entrée), du « L2-norm pooling ». Dans les faits, même si initialement l'average pooling était souvent utilisé il s'est avéré que le max-pooling était plus efficace car celui-ci augmente plus significativement l'importance des activations fortes. En d'autres circonstances, on pourra utiliser un pooling stochastique.

Le pooling permet de gros gains en puissance de calcul. Cependant, en raison de la réduction agressive de la taille de la représentation (et donc de la perte d'information associée), la tendance actuelle est d'utiliser de petits filtres (type 2x2). Il est aussi possible d'éviter la couche de pooling mais cela implique un risque sur-apprentissage plus important.[23

I.12.3 Couches de correction (RELU) :

Il est possible d'améliorer l'efficacité du traitement en intercalant entre les couches de traitement une couche qui va opérer une fonction mathématique (fonction d'activation) sur les signaux de sortie.

La fonction ReLU (abréviation de Unités Rectifié linéaires) : F(x)=max(0,x) Cette fonction force les neurones à retourner des valeurs positives.[23]

I.12.4 Couche entièrement connectée (FC) :

Après plusieurs couches de convolution et de max-pooling, le raisonnement de haut niveau dans le réseau neuronal se fait via des couches entièrement connectées. Les neurones dans une couche entièrement connectée ont des connexions vers toutes les sorties de la couche précédente. Leurs fonctions d'activations peuvent donc être calculées avec une multiplication matricielle suivie d'un décalage de polarisation.[23]

I.12.5 Couche de perte (LOSS) :

La couche de perte spécifie comment l'entrainement du réseau pénalise l'écart entre le signal prévu et réel. Elle est normalement la dernière couche dans le réseau. Diverses fonctions de perte adaptées à différentes tâches peuvent y être utilisées. La fonction « Softmax » permet de calculer la distribution de probabilités sur les classes de sortie.[23]

Exemples de modèles de CNN :

La forme la plus commune d'une architecture CNN empile quelques couches Conv-ReLU, les suit avec des couches Pool, et répète ce schéma jusqu'à ce que l'entrée soit réduite dans un espace d'une taille suffisamment petite. À un moment, il est fréquent de placer des couches entièrement connectées (FC). La dernière couche entièrement connectée est reliée vers la sortie. Voici quelques architectures communes CNN qui suivent ce modèle (figure10):

- INPUT -> CONV -> RELU -> FC

- INPUT -> [CONV -> RELU -> POOL] * 2 -> FC -> RELU -> FC Ici, il y a une couche de CONV unique entre chaque couche POOL

- INPUT -> [CONV -> RELU -> CONV -> RELU -> POOL] * 3 -> [FC -> RELU] * 2 -> FC Ici, il y a deux couches CONV empilées avant chaque couche POOL.

L'empilage des couches CONV avec de petits filtres de pooling (plutôt un grand filtre) permet un traitement plus puissant, avec moins de paramètres. Cependant, avec l'inconvénient de demander plus de puissance de calcul (pour contenir tous les résultats intermédiaires de la couche CONV)[23].

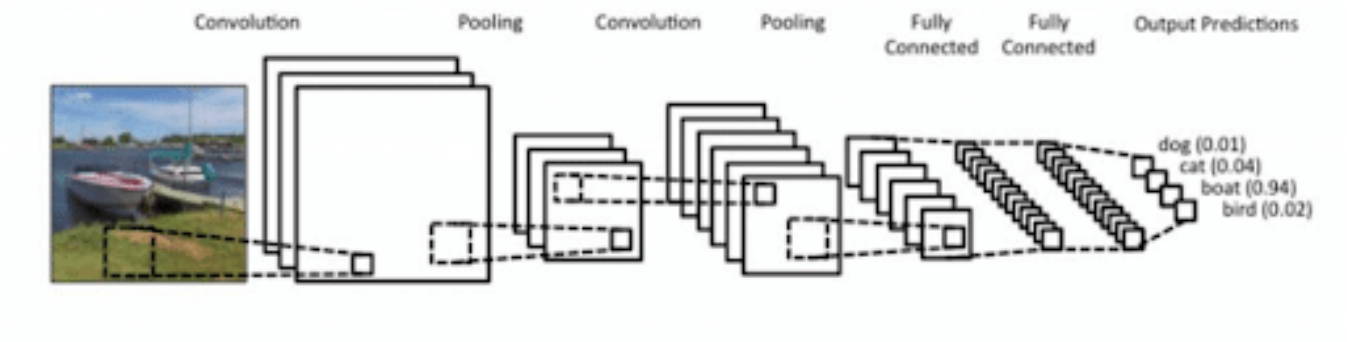


Figure 10 : Les différentes couches que constitue un cnn [23]

I.13 Réseaux de neurones récurrents :

Le réseau neuronal récurrent (RNN) est un type de réseau neuronal où la sortie de l'étape précédente est alimentée en entrée de l'étape actuelle . Dans les réseaux de neurones traditionnels, toutes les entrées et sorties sont indépendantes les unes des autres, mais dans des cas comme lorsqu'il est nécessaire de prédire le mot suivant d'une phrase, les mots précédents sont nécessaires et il est donc nécessaire de se souvenir des mots précédents. Ainsi, RNN a vu le jour, ce qui a résolu ce problème à l'aide d'une couche cachée. La caractéristique principale et la plus importante de RNN est l' état caché , qui mémorise certaines informations sur une séquence.[24]

I.14 Conclusion :

Dans ce chapitre nous avons présenté l'état de l'art des méthodes utilisées pour la réalisation de notre projet ainsi que de diverses généralités reliées à notre sujet.

Chapitre II :

Conception du modèle

Introduction :

Dans ce chapitre nous allons nous intéresser en détail sur l'architecture de notre réseau de neurones profond ainsi que la base de données utilisée et ses caractéristiques avec les couches présentes dans notre modèle.

II.1 Flot de conception :

Afin d'être traitée et reconnue, notre image passe par les étapes suivantes (figure 11) :

Acquisition de l'image :

C'est là que nous allons alimenter notre réseau de neurones avec une image de texte.

***Prétraitment* :**

C'est l'étape ou se fera le redimensionnement de l'image et ou se normalise les valeurs de gris de l'image ce qui simplifie la tâche pour notre réseau de neurones

***Segmentation* :**

Cela se fera grâce a un algorithme a part dont nous parlerons plus tard, cette étape sert à segmenter une seule image en plusieurs images dont chacune d'entre elles contiendra un mot.

Extraction de caractéristiques :

C'est l'étape qui contient les couches de convolutions et de pooling ainsi que des filtres.

***Classification* :**

Cette étape servira a prédire la vraisemblance ou la probabilité que les données qui suivent tomberont dans l'une des catégories prédéterminées

Sortie/Résultat :

La sortie de notre modèle et l'affichage de notre texte après traitement.

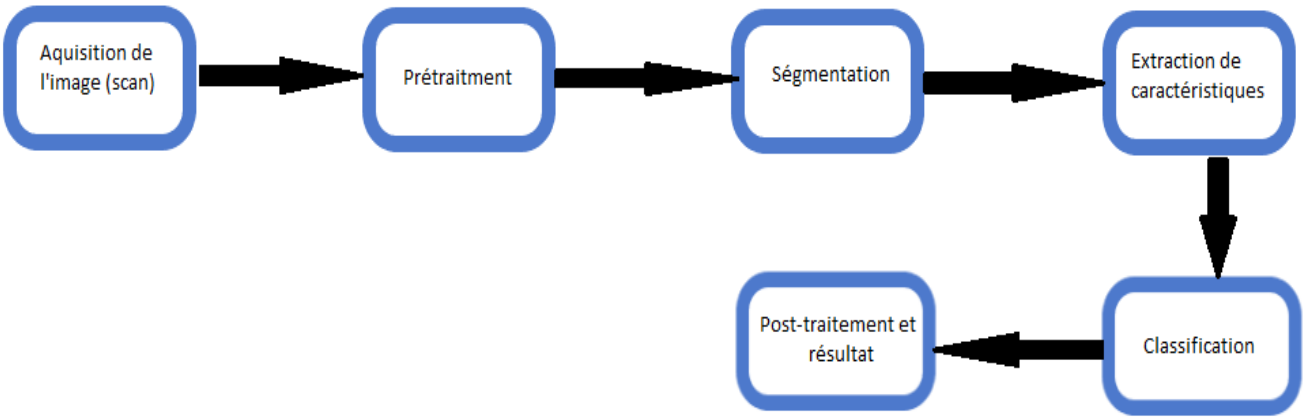


Figure 11:Schéma synoptique des étapes de traitement que passera par notre image d'éntrée

II.2 La base de données :

Lorsque que l'on crée un modèle de réseaux de neurones convolutifs ou tout simplement quand on utilise un modèle déjà existant, nous devons entraîner ce modèle avec des données afin qu'il puisse apprendre et être capable de reconnaître un objet ou quelque chose en particulier. Pour cela on utilise une base de donnée contenant toutes nos images, dans notre cas on utilisera la base de donnée d'écriture manuscrite IAM.

Une partie de cette base de données servira à entraîner le modèle CNN, c'est ce qu'on appelle «la base de données d'entraînement». La partie de la base de données restante sert pour évaluer le modèle, on l'appelle la «base de données de validation».

II.2.1 Aperçu de la base de donnée IAM :

La base de données d'écriture manuscrite IAM contient des formes de texte anglais manuscrit qui peuvent être utilisées pour former et tester des outils de reconnaissance de texte manuscrit et pour effectuer des expériences d'identification et de vérification de l'auteur.

La base de données contient des formes de texte manuscrit sans contrainte, qui ont été numérisées à une résolution de 300 dpi et enregistrées sous forme d'images PNG avec 256 niveaux de gris. Les figure ci-dessous (figures 12,13,14) fournit des exemples d'un formulaire complet, d'une ligne de texte et de quelques mots extraits.

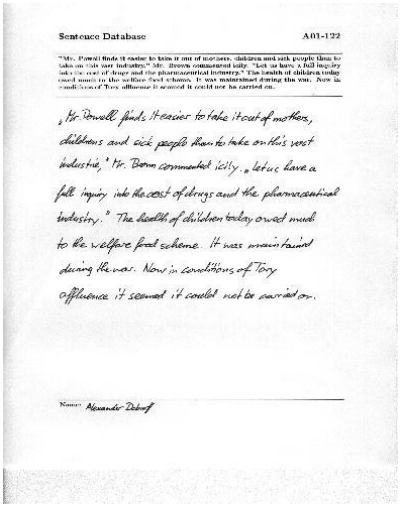


Figure 12 : Image de la base de données IAM contenante un texte

Figure 13 Image de la base de données IAM contenante une phrase

Figure 14 Image de la base de données IAM contenante un mot

Tous les formulaires ainsi que toutes les lignes de texte extraites, les mots et les phrases peuvent être téléchargés sous forme de fichiers PNG, avec les méta-informations XML correspondantes incluses dans les fichiers image.

Les caractéristiques de la base de données IAM :

-La base de données d'écriture manuscrite IAM 3.0 est structurée comme suit :

-657 écrivains ont fourni des échantillons de leur écriture

-1'539 pages de texte scanné

-5'685 phrases isolées et étiquetées

-13'353 lignes de texte isolées et étiquetées

-115'320 mots isolés et étiquetés [25]

II.3 Classification temporelle connexionniste :

La classification temporelle connexionniste (CTC) est un type de sortie de réseau de neurones utile pour résoudre les problèmes de séquence tels que l'écriture manuscrite et la reconnaissance vocale où la synchronisation varie. L'utilisation de CTC garantit que l'on n'a pas besoin d'un ensemble de données aligné, ce qui rend le processus de formation plus simple. (figure15)
On va utiliser cette technique comme dernière couche dans notre réseau de neurones.[26]

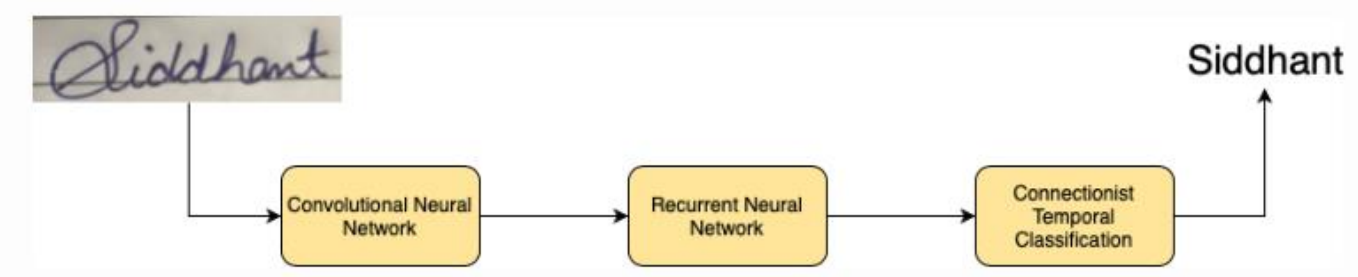


Figure 15 : Flux de reconnaissance de l'écriture manuscrite à l'aide de CNN et RNN [26]

II.3.1 Rôle de la classification temporelle connexionniste :

Dans le cas de la création d'un OCR (Optical Character Reader), les CRNN (Convolutional Recurrent Neural Networks) sont un choix privilégié. Ils produisent un score de caractère pour chaque pas de temps, qui est représenté par une matrice. Nous devons maintenant utiliser cette matrice pour :
-Former le réseau de neurones, c'est-à-dire calculer la perte
-Décodage de la sortie du réseau de neurones
Le fonctionnement CTC aide à réaliser les deux tâches.

II.3.2 Problèmes résolus par la classification temporelle connexionniste :

Si on crée un dataset rempli d'images de texte et que nous spécifions chaque pas de temps du caractère correspondant de l'image. Cette approche présente quelques problèmes :
-Annoter un dataset au niveau des caractères est une tâche fastidieuse.
-Il en résultera une duplication des caractères si le personnage prend plus d'un pas de temps.[26]

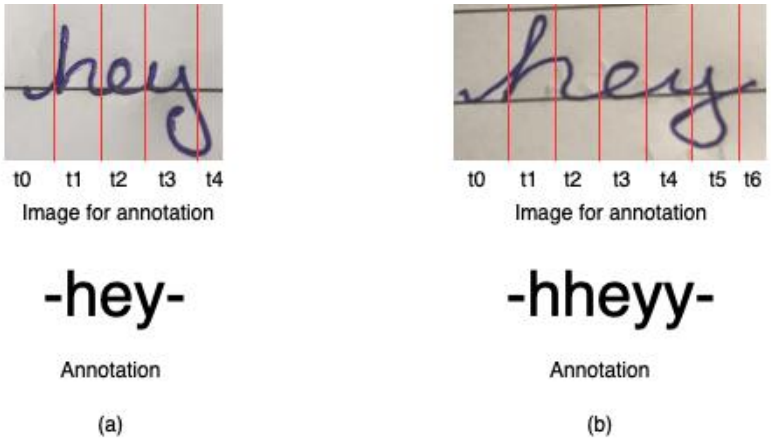


Figure 16 : les différentes annotations qu'occupe les étapes uniques [26]

Ici, la classification temporelle connexionniste nous viendra en aide :

-la classification temporelle connexionniste est formulé de telle manière qu'il ne nécessite que le texte qui apparaît dans l'image. Nous pouvons ignorer à la fois la largeur et la position des caractères dans une image.

- Il n'est pas nécessaire de post-traiter la sortie de l'opération de la classification temporelle connexionniste , En utilisant des techniques de décodage, nous pouvons obtenir directement le résultat du réseau.

II.3.3 Fonctionnement de la classification temporelle connexionniste :

La CTC travaille sur les trois grands concepts suivants :

-Encodage du texte

-Calcul des pertes

- Décodage

Encodage du texte :

Que faire quand le caractère prends plus d'un ot dans l'image est "hey" où "h" prend trois pas de temps, "e" et "y" prennent un pas de tempas de temps dans l'image, Les méthodes non-CTC échoueraient ici et donneraient des caractères en double. Pour résoudre ce problème, la CTC fusionne tous les caractères répétés en un seul caractère. Par exemple, si le mps chacun. Ensuite, la sortie du réseau utilisant CTC sera "hhhey", qui, selon notre schéma d'encodage, est réduite à "hey". Maintenant cette question se pose : qu'en est-il des mots où il y a des caractères qui se répètent ?

Pour gérer ces cas, CTC introduit un pseudo-caractère appelé blanc noté "-" dans les exemples suivants.

Lors de l'encodage du texte, si un caractère se répète, un espace est placé entre les caractères dans le texte de sortie. Considérons le mot 'meet', les codages possibles seront 'mm-ee-ee-t', 'mmm-e-e-ttt', le mauvais

codage sera 'mm-eee-tt', car il en résultera dans 'met' lorsqu'il est décodé. Le CRNN est formé pour produire le texte codé.

Calcul des pertes :

Pour former le CRNN, nous devons calculer la perte en fonction de l'image et de son étiquette. Nous obtenons une matrice du score pour chaque personnage à chaque pas de temps du CRNN. La figure (…) montre un exemple de matrice de sortie du CRNN. Il y a trois pas de temps et trois caractères (dont un blanc). A chaque pas de temps, le score du personnage totalise 1.

Pour calculer la perte, tous les scores des alignements possibles de la vérité terrain sont additionnés. De cette manière, l'emplacement du personnage dans l'image n'est pas significatif.

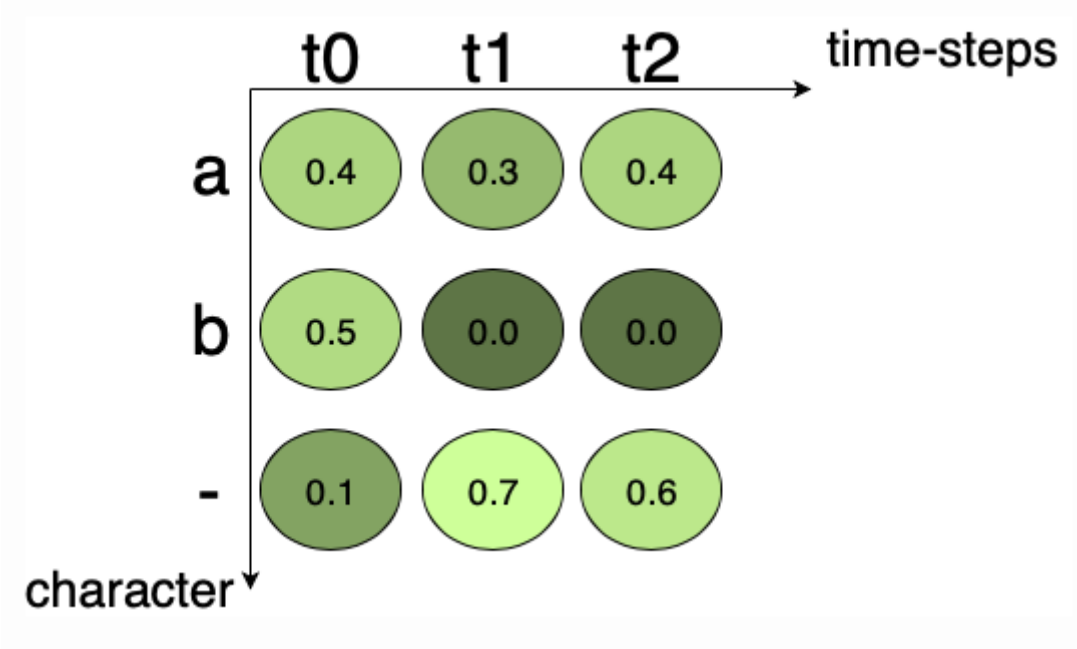


Figure17 : Matrice de sortie du réseau de neurones montrant la probabilité à chaque pas de temps [26]

Les scores des caractères correspondants sont multipliés pour obtenir le score d'un chemin. Dans la Figure.(17), le score pour le chemin "a–" est 0,4x0,7x0,6 = 0,168, et pour le chemin "aaa" est 0,4x0,3x0,4 = 0,048. Pour obtenir le score correspondant à la vérité terrain donnée, les scores de tous les chemins vers le texte correspondant sont additionnés. Par exemple, si la vérité terrain est "a", tous les chemins possibles pour "a" dans la Fig(…) sont "aaa", "a–", "a-", "aa-", "-aa", " -un". En résumant le score du chemin individuel que nous obtenons, 0,048 + 0,168 + 0,018 + 0,072 + 0,012 + 0,028 = 0,346. 0,346 est la probabilité que la vérité terrain se produise, ce n'est pas la perte.

Comme nous le savons, la perte est le logarithme négatif de la probabilité, elle peut être calculée facilement. Cette perte peut être rétro-propagée et le réseau peut être formé.

Décodage *:*

Une fois que CRNN est formé, nous voulons qu'il nous donne une sortie sur des images de texte invisibles. En d'autres termes, nous voulons le texte le plus probable étant donné une matrice de sortie du CRNN. Une méthode peut consister à examiner chaque sortie de texte potentielle, mais ce ne sera pas très pratique du point de vue du calcul. L'algorithme du meilleur chemin est utilisé pour surmonter ce problème.

Il se compose des deux étapes suivantes :

-Calculer le meilleur chemin en considérant le caractère avec la probabilité maximale à chaque pas de temps.

-Cette étape consiste à supprimer les blancs et les caractères en double, ce qui donne le texte réel.

Par exemple, considérons la matrice de la Fig.(…). Si nous considérons le premier pas de temps t0, alors le caractère avec la probabilité maximale est « b ». Pour t1 et t2, le caractère avec la probabilité maximale est '-' et '-' respectivement. Ainsi, le texte de sortie selon l'algorithme du meilleur chemin pour la matrice de la Fig.(…) après décodage est "b". [26]

II.4 Présentation du modèle :

Nous utilisons un réseau de neurones pour notre tâche. Il se compose de couches de réseaux de neurones convolutifs (CNN), de couches de réseaux de neurones récurrents (RNN) et d'une couche finale de classification temporelle connexionniste (CTC).

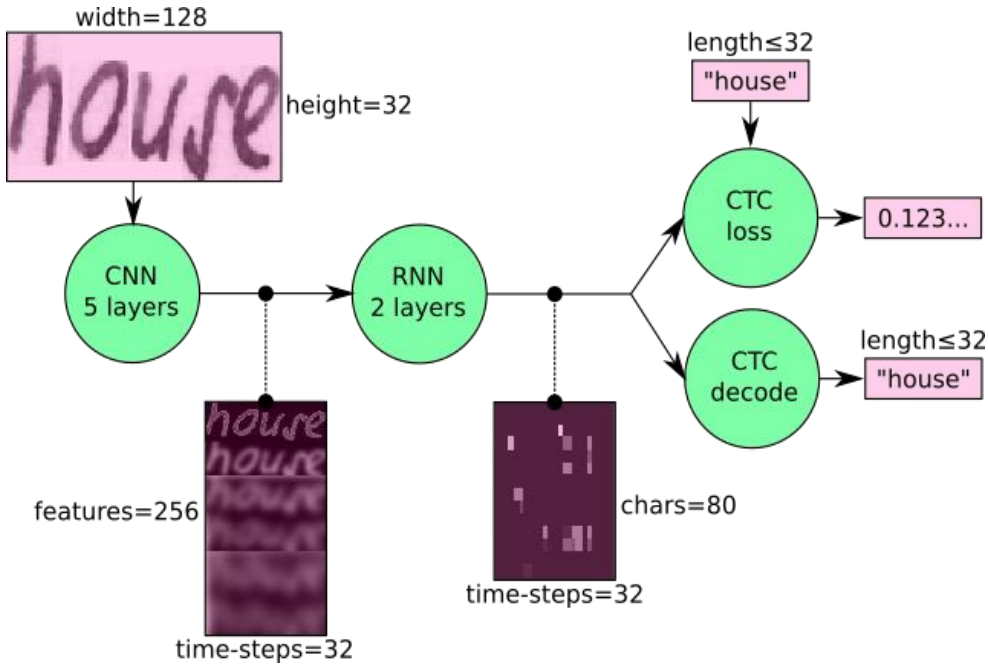


Figure18 : aperçu de notre réseau de neurones[27]

Nous pouvons également voir le réseau de neurones de manière plus formelle comme une fonction qui mappe une image (ou matrice) M de taille W×H à une séquence de caractères (c1, c2, …) avec une longueur comprise entre 0 et L.

Le texte est reconnu au niveau des caractères, donc les mots ou les textes non contenus dans les données d'apprentissage peuvent également être reconnus (tant que les caractères individuels sont correctement classés).

$$\underset{W\times H}{M} \rightarrow \underset{0\leq n\leq L}{(c_1, c_2, \ldots, c_n)}$$

Opérations :

CNN : l'image d'entrée est introduite dans les couches CNN. Ces couches sont formées pour extraire les caractéristiques pertinentes de l'image. Chaque couche se compose de trois opérations. Premièrement, l'opération de convolution, qui applique un noyau de filtre de taille 5 × 5 dans les deux premières couches et 3 × 3 dans les trois dernières couches à l'entrée. Ensuite, la fonction RELU non linéaire est appliquée. Enfin, une couche de regroupement résume les régions d'image et génère une version réduite de l'entrée. Alors que la hauteur de l'image est réduite de 2 dans chaque couche, des cartes d'entités (canaux) sont ajoutées, de sorte que la carte d'entités (ou séquence) de sortie a une taille de 32 × 256.

RNN : la séquence de caractéristiques contient 256 caractéristiques par pas de temps, le RNN propage les informations pertinentes à travers cette séquence. L'implémentation populaire de la mémoire à long court terme (LSTM) des RNN est utilisée, car elle est capable de propager des informations sur de plus longues distances et fournit des caractéristiques de formation plus robustes que la RNN classique. La séquence de sortie RNN est mappée sur une matrice de taille 32 × 80. L'ensemble de données IAM se compose de 79 caractères différents, un autre caractère supplémentaire est nécessaire pour l'opération CTC, il y a donc 80 entrées pour chacun des 32 pas de temps.

CTC : lors de la formation du Réseau de neurones, le CTC reçoit la matrice de sortie RNN et le texte de vérité terrain et calcule la valeur de perte. Lors de l'inférence, le CTC ne reçoit que la matrice et il la décode dans le texte final. Le texte de vérité terrain et le texte reconnu peuvent comporter au maximum 32 caractères.[27]

II.4.1Donnée utilisée :

Entrée : il s'agit d'une image en niveaux de gris de taille 128×32. Habituellement, les images de base de données n'ont pas exactement cette taille, donc nous la redimensionnons (sans distorsion) jusqu'à ce qu'elle ait une largeur de 128 ou une hauteur de 32 pixels. Ensuite, nous copions l'image dans une image cible (blanche) de taille 128×32. Ce processus est illustré à la Fig(19). Enfin, nous normalisons les niveaux de gris de l'image, ce qui simplifie la tâche pour le réseau de neurones. L'augmentation des données peut facilement être intégrée en copiant l'image dans des positions aléatoires au lieu de l'aligner à gauche ou en redimensionnant l'image de manière aléatoire.

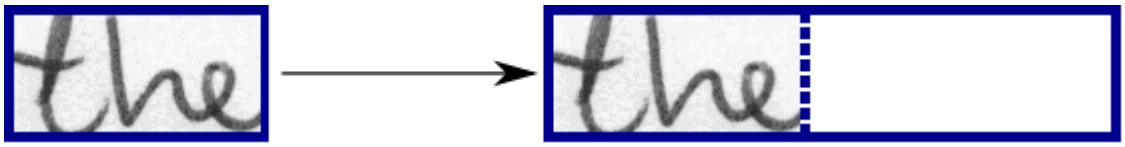


FIGURE 19 : Une image de la base de donnée avec une taille normale avant son adaptation vers la taille 128*32

Sortie CNN : la figure 20 montre la sortie des couches CNN qui est une séquence de longueur 32. Chaque entrée contient 256 caractéristiques. Bien sûr, ces caractéristiques sont ensuite traitées par les couches RNN, cependant, certaines d'éntre eux montrent déjà une forte corrélation avec certaines propriétés de haut niveau de l'image d'entrée : il existe des caractéristiques qui ont une forte corrélation avec les caractères (par exemple "e"), ou avec des caractères en double (par exemple "tt"), ou avec des propriétés de caractère telles que des boucles (contenues dans des "l" ou des "e" manuscrits).[27]

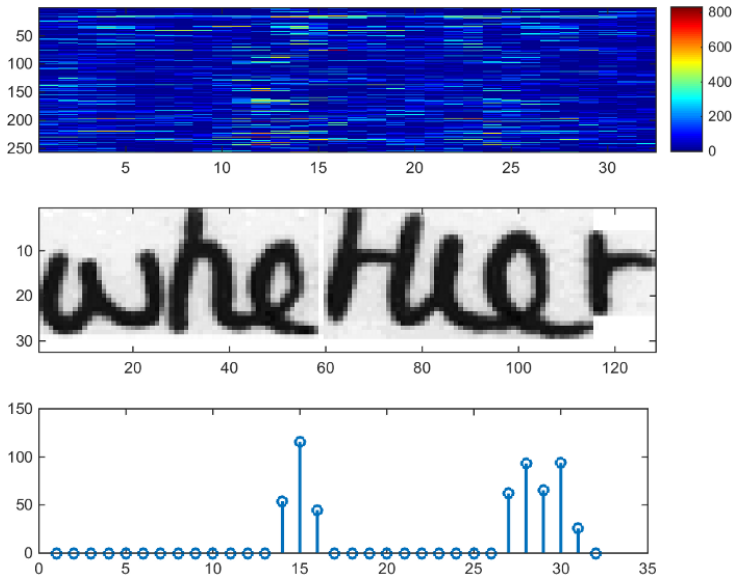


Figure 20 : En haut : 256 caractéristiques par pas de temps sont calculées par les couches CNN. Milieu : image d'entrée. En bas : tracé de la 32e caractéristique, qui présente une forte corrélation avec l'occurrence du caractère « e » dans l'image. [27]

Sortie RNN : la figure (21) montre une visualisation de la matrice de sortie RNN pour une image contenant le texte "petit". La matrice affichée dans le graphique le plus haut contient les scores des caractères, y compris l'étiquette vierge CTC comme dernière (80e) entrée. Les autres entrées de la matrice, de haut en bas, correspondent aux caractères suivants :

" !"#&'()*+,-./0123456789:;?ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz".

On peut voir que la plupart du temps, les caractères sont prédits exactement à la position où ils apparaissent dans l'image (par exemple, comparer la position du "i" dans l'image et dans le graphique). Seul le dernier caractère « e » n'est pas aligné. Mais c'est normale, car l'opération CTC est sans segmentation et ne se soucie pas des positions absolues. À partir du graphique le plus bas montrant les scores pour les caractères "l", "i", "t", "e" et l'étiquette vierge CTC, le texte peut facilement être décodé : nous prenons simplement le caractère le plus probable à chaque étape, cela forme le soi-disant meilleur chemin, puis nous supprimons les caractères répétés et enfin tous les blancs : "l---ii--t-t--l-…-e" → "l---i--t-t- -l-…-e » → « peu ».[27]

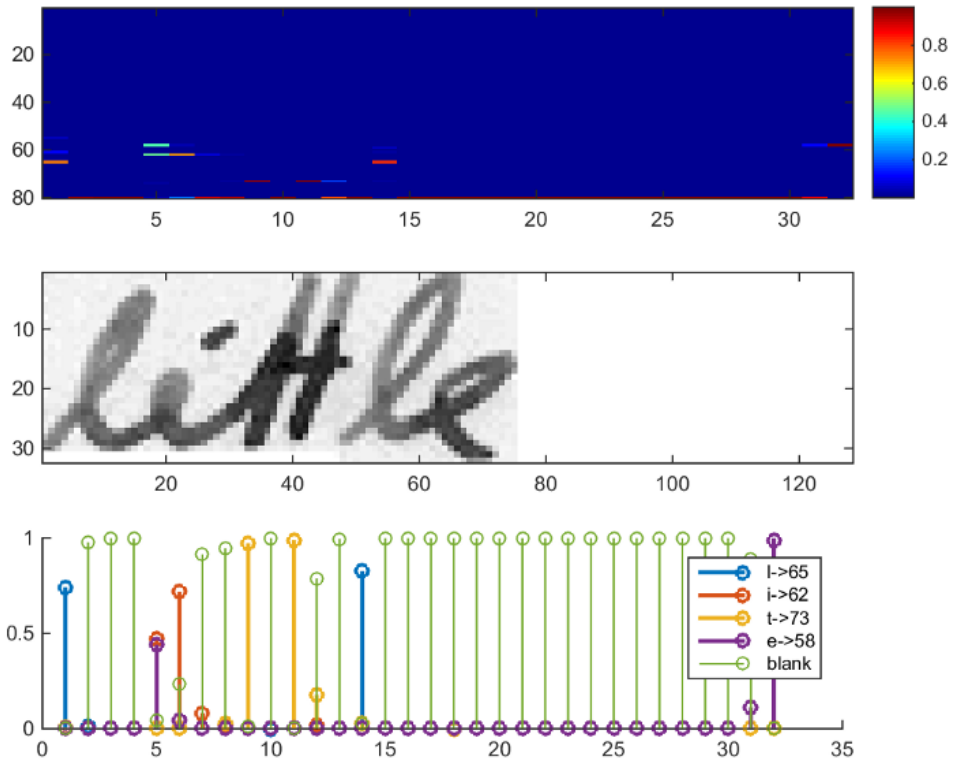


Figure 21 : En haut : matrice de sortie des couches RNN. Milieu : image d'entrée. En bas : Probabilités pour les caractères « l », « i », « t », « e » et l'étiquette vierge CTC. [27]

II.5 Conclusion :

Nous avons expliqué dans ce chapitre l'architecture de notre réseau de neurones utilisé ainsi que les étapes que passe par notre image d'entrée et les couches responsables au traitement de cette dernière tout en donnant les caractéristiques de notre base de données IAM.

Chapitre III :

Implémentation et résultats

Introduction :

Dans ce chapitre nous plongerons dans les caractéristiques de notre réseau de neurones ainsi que l'exécution de notre apprentissage, nous expliquerons en détail tous les outils utilisés ainsi que l'implémentation des algorithmes conçus pour réaliser notre programme de reconnaissance de texte manuscrit, et nous présenterons nos résultats.

Figure 22 :exemple de sortie d'un mot manuscrit

III.1 Logiciels et librairies utilisés dans l'implémentation :

III.1.1 python :

Langage de programmation utilisé en machine learning et en data science, le langage Python s'impose également dans d'autres secteurs d'activité grâce à sa simplicité et sa compatibilité.

Le langage Python est un langage de programmation open source multi-plateformes et orienté objet. Grâce à des bibliothèques spécialisées, Python s'utilise pour de nombreuses situations comme le développement logiciel, l'analyse de données, ou la gestion d'infrastructures.

Langage de programmation interprété, Python permet l'exécution du code sur n'importe quel ordinateur. Utilisable aussi bien par des programmeurs débutants qu'experts.[28]

III.1.2 pycharm :

PyCharm est un environnement de développement intégré utilisé pour programmer en Python.Il permet l'analyse de code et contient un débogueur graphique. Il permet également la gestion des tests unitaires, l'intégration de logiciel de gestion de versions, c'est le meilleur IDE qui nous aidera a réaliser notre réseau de neurones.[29]

III.1.3 Tensorflow :

Il s'agit d'une bibliothèque Open Source de calcul numérique et de Machine Learning compatible avec le langage Python. Elle simplifie le processus d'acquisition de données, d'entraînement des modèles de Machine Learning, de génération de prédictions et de raffinement des résultats futurs.

TensorFlow regroupe des modèles et des algorithmes de Machine Learning et de Deep Learning.

Il est possible de se servir de ce framework pour entraîner et exécuter des réseaux de neurones profonds pour la classification de chiffres manuscrits, pour la reconnaissance d'image, pour le plongement lexical, pour les réseaux de neurones récurrents, pour les modèles " sequence-to-sequence " de traduction automatique, pour le traitement naturel du langage, et pour les simulations basées sur les équations aux dérivées partielles.[30]

III.1.4 Keras :

Keras est une API de réseau de neurones écrite en langage Python. Il s'agit d'une bibliothèque Open Source, exécutée par-dessus des frameworks tels que TensorFlow.

Conçue pour être modulaire, rapide et simple d'utilisation, Keras a été créée par l'ingénieur François Chollet de Google. Elle offre une façon simple et intuitive de créer des modèles de Deep Learning.

Un Modèle Keras est constitué d'une séquence ou d'un graphique indépendant. Il existe plusieurs modules entièrement configurables et pouvant être combinés pour créer de nouveaux modèles.

L'un des avantages de cette modularité est qu'il est très facile d'ajouter de nouvelles fonctionnalités sous forme de modules séparés. Keras est donc très flexible, et adapté à la recherche et à l'innovation.[31]

III.1.5 Edit distance :

Mise en œuvre rapide de la distance d'édition (distance de Levenshtein).

Cette bibliothèque implémente simplement la distance de Levenshtein avec C++ et Cython.[32]

III.1.6 Open cv :

OpenCV est une bibliothèque Python utilisée pour résoudre des problèmes de vision par ordinateur.

La vision par ordinateur comprend la compréhension et l'analyse d'images numériques par l'ordinateur et le traitement des images ou la fourniture de données pertinentes après analyse de l'image[33]

III.1.7 Matplotlib :

Matplotlib est une bibliothèque multiplateforme de visualisation de données et de traçage graphique pour Python et son extension numérique NumPy. En tant que tel, il offre une alternative open source viable à MATLAB. Les développeurs peuvent également utiliser les API (Application Programming Interfaces) de matplotlib pour intégrer des tracés dans des applications GUI.[34]

III.1.8 NumPy :

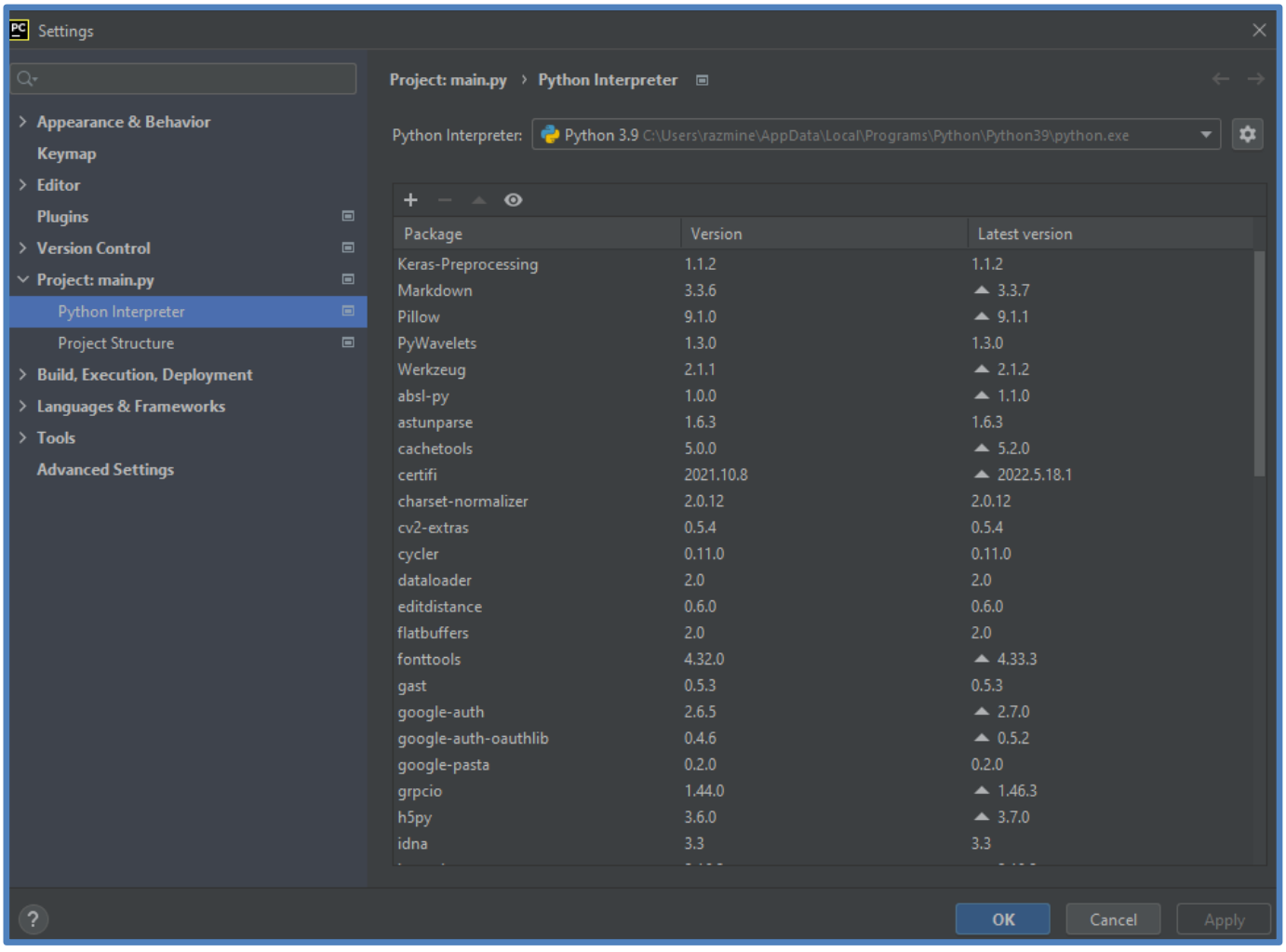
NumPy, qui signifie Numerical Python, est une bibliothèque composée d'objets de tableau multidimensionnel et d'une collection de routines pour traiter ces tableaux. En utilisant NumPy, des opérations mathématiques et logiques sur des tableaux peuvent être effectuées.[35]

III.1.9 Path :

Ce module propose des classes représentant des chemins de système de fichiers avec une sémantique appropriée pour différents systèmes d'exploitation. Les classes de chemin sont divisées entre les chemins purs, qui fournissent des opérations purement informatiques sans E/S, et les chemins concrets, qui héritent des chemins purs mais fournissent également des opérations d'E/S.[36]

Maintenant on va montrer comment installer ces bibliothèques :

-On installe et importe les bibliothèques a partir de pycharm (file->settings->Python interpreter) :



Si on ne trouve pas la bibliothèque, sois on la télécharge a partir d'internet ou bien on la télécharge a partir de l'invite de commande en utilisant l'outil pip :

III.2 Environnement de notre travail :

En raison de la faible puissance de traitement de notre processeur et du manque de mémoire vive (8Gb étant pas suffisante pour réaliser notre tache) de notre système, l'exécution d'une tâche de ce calibre (entrainement du dataset) prendrait des jours sans éteindre notre système ce qui le ferait surchauffer et ralentirait encore plus le processus.

Nous avons donc opté pour utiliser un autre ordinateur plus puissant qui réduirait considérablement le temps d'apprentissage (de quelques jours a seulement quelques heures)

Nous allons donc faire l'apprentissage sur notre puissant ordinateur et ensuite le transférer vers notre deuxième ordinateur pour faire l'exécution.

Voici la configuration de nos deux machines :

Machine dédiée à l'apprentissage :

-Macbook Pro Avec Intel® Core™ I7 Processor 2.6 GHz (6cores /12 threads)

-16Gb de mémoire vive

-256 de stockage SSD

Et pour la machine dédié à l'exécution :

Intel® Core™ i5-7200U Processor 2.5 GHz (3M Cache, up to 3.1 GHz)

NVIDIA® GeForce® 920MX (N16V-GMR1)

8Gb de mémoire vive @ 2.133 Ghz

256GB de stockage SATA SSD 2.5''

III.3 Exécution de notre apprentissage :

On télécharge la base de données a partir de son site. [38]

On précise que notre base de données sera divisée en deux catégories :

95% Données d'apprentissage : qui aidera à former notre réseau de neurones.

5% données de validation : ceux sont les données qui seront utilisées pour valider les performances de notre modèle lors de la formation. Ce processus de validation donne des informations qui nous aideront a prendre des notes sur l'état de notre taux d'erreur de caractère ainsi que la précision des mots après chaque époque.

-Après téléchargement, on crée deux répertoires dans notre ordinateur **img** et **gt**.

-On met le documents **words.txt** dans le répertoire **gt.**

-On prend ensuite le contenu de notre base de données dans le répertoire **img.**

-On ouvre notre terminal a partir du répertoire source, et on lance l'apprentissage avec la commande suivante : **python main.py –mode train –data_dir /Users/kaos/Documents/tensorflow/dataset/IAM(figure ci-dessous)**

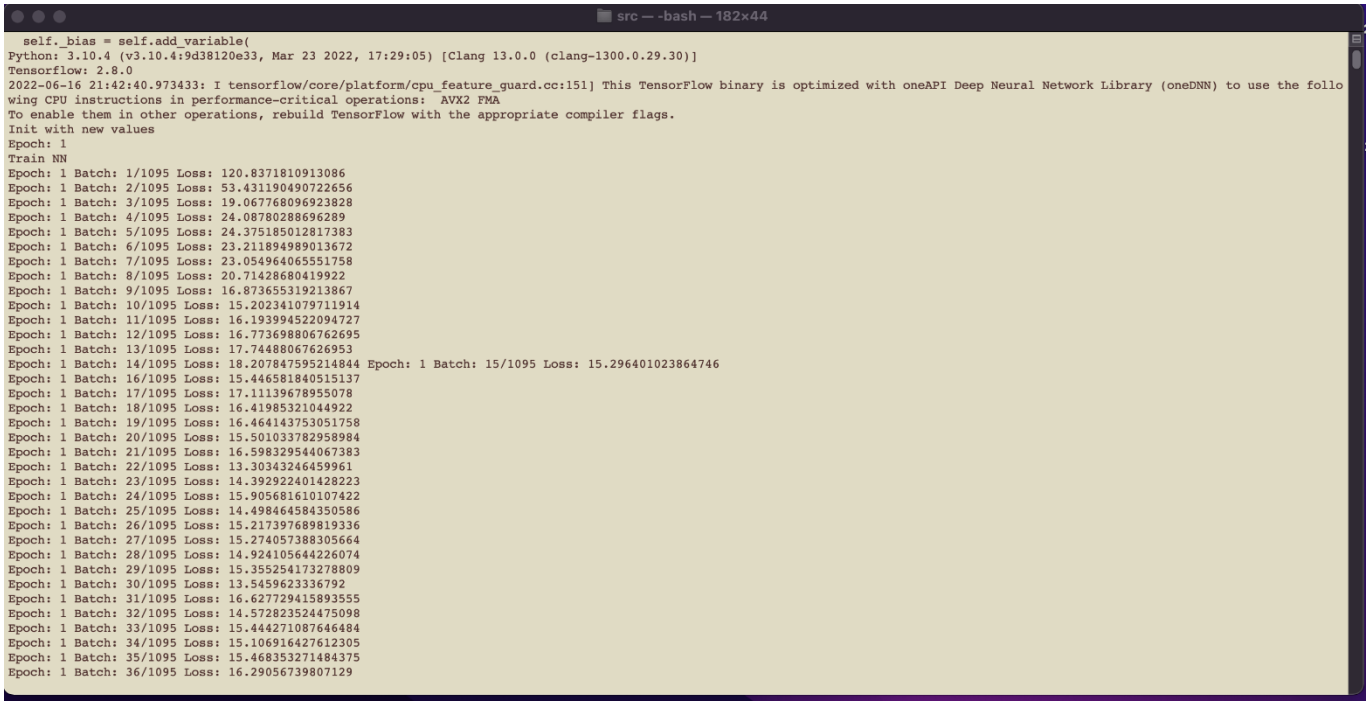
Figure 23 : exécution de notre apprentissage

Immédiatement après, notre apprentissage commence sachant que notre programme est réalisé de manière a ce qu'il continue l'apprentissage tant que le taux d'erreur de caractère s'améliore d'époch après époch

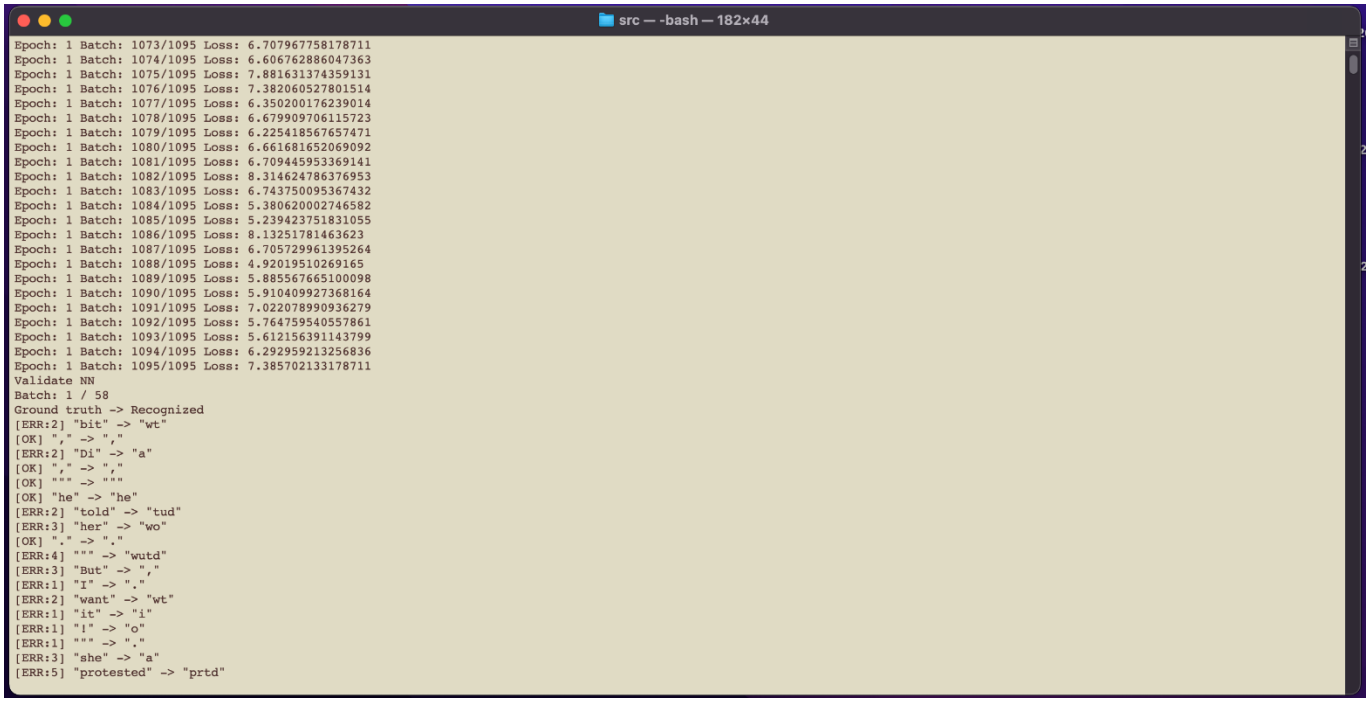
et ne s'arrêtera qu'après 25 (justification) époques consécutives sans amélioration du taux d'erreur de caractère.

Notre programme vas balayer et traiter les images de notre base de données et répéter ce processus après chaque époque afin d'améliorer le taux d'érreur.

On note aussi que chaque époque consiste de 1095 lots d'apprentissage et de 58 lots de validation. Après la fin de chaque époque, commence la validation.



Après la fin de chaque époque, commence la validation.



Notre programme générera le taux d'erreur de caractère ainsi que la précision des mots comme statistiques après la fin de chaque validation, par exemple pour la première validation notre taux d'erreur est de 45.66% et la précision des mots est de 31.06% , qui n'est pas un bon résultat ce qui voudrait dire qu'il va entamer la deuxième époque

Les résultats de la deuxième validation est donné ci-dessous :

```
Character error rate: 25.641824249165748. Word accuracy: 49.1155046826222668.
Character error rate improved, save model
```

Nous cherchons le taux le plus bas possible pour le taux d'erreur de caractère et un taux élevé pour la précision des mots, on remarque une belle amélioration de notre TEC ainsi que la précision des mots après la deuxième époque.

Notre programme entame la troisième époque :

```
Epoch: 3
Train NN
Epoch: 3 Batch: 1/1095 Loss: 3.7133755683898926
Epoch: 3 Batch: 2/1095 Loss: 4.5572638511657715
Epoch: 3 Batch: 3/1095 Loss: 3.4728734493255615
Epoch: 3 Batch: 4/1095 Loss: 4.420389175415039
Epoch: 3 Batch: 5/1095 Loss: 3.118698835372925
Epoch: 3 Batch: 6/1095 Loss: 3.020918607711792
Epoch: 3 Batch: 7/1095 Loss: 2.6757020950317383
Epoch: 3 Batch: 8/1095 Loss: 3.6498425006866455
Epoch: 3 Batch: 9/1095 Loss: 4.222884654998779
Epoch: 3 Batch: 10/1095 Loss: 4.041409969329834
Epoch: 3 Batch: 11/1095 Loss: 3.708270311355591
Epoch: 3 Batch: 12/1095 Loss: 3.1600029468536377
Epoch: 3 Batch: 13/1095 Loss: 3.199570655822754
Epoch: 3 Batch: 14/1095 Loss: 4.336920261383057
Epoch: 3 Batch: 15/1095 Loss: 3.5806024074554443
Epoch: 3 Batch: 16/1095 Loss: 3.252180814743042
```

Résultat de la validation de notre troisième époque :

```
Character error rate: 20.329254727474497%. Word accuracy: 56.0006937218175555%.
Character error rate improved, save model
Epoch: 4
Train NN
Epoch: 4 Batch: 1/1095 Loss: 2.8737518787384033
Epoch: 4 Batch: 2/1095 Loss: 3.4005565643310547
Epoch: 4 Batch: 3/1095 Loss: 3.6581802368164062
Epoch: 4 Batch: 4/1095 Loss: 3.418830633163452
```

On remarque toujours une amélioration considérable.

on saute vers la quinzième époque :

```
Character error rate: 11.701890989988877%. Word accuracy: 71.488033298647255%.
Character error rate improved, save model
Epoch: 15
Train NN
Epoch: 15 Batch: 1/1095 Loss: 0.8533678650856018
Epoch: 15 Batch: 2/1095 Loss: 1.1979608535766602
Epoch: 15 Batch: 3/1095 Loss: 1.1407861709594727
Epoch: 15 Batch: 4/1095 Loss: 1.4222582578659058
```

On remarque de bons résultats sachant que le TEC touche les 11% et notre précision des mots est a 71.48%.

Vers la 30ème époque on obtiens les résultats suivants :

```
Character error rate: 11.5328142380422695%. Word accuracy: 71.713492889351375%.
Character error rate not improved, best so far: 11.5328142380422695%
Epoch: 30
Train NN
Epoch: 30 Batch: 1/1095 Loss: 1.1104263067245483
Epoch: 30 Batch: 2/1095 Loss: 0.7681021094322205
Epoch: 30 Batch: 3/1095 Loss: 0.8977437019348145
Epoch: 30 Batch: 4/1095 Loss: 0.8054177761077881
Epoch: 30 Batch: 5/1095 Loss: 0.615505576133728
```

Le TEC et notre précision des mots commence a stagner ce qui veut dire que notre réseau de neurones a fait un très bon progrès.

15 époques plus tard, voila nos résultats :

```
Character error rate: 10.745272525027815%. Word accuracy: 73.413111342351735%.
Character error rate improved, save model
Epoch: 45
Train NN
Epoch: 45 Batch: 1/1095 Loss: 0.40757986903190613
Epoch: 45 Batch: 2/1095 Loss: 0.46913033723831177
Epoch: 45 Batch: 3/1095 Loss: 1.1053404808044434
Epoch: 45 Batch: 4/1095 Loss: 0.2519353926181793
Epoch: 45 Batch: 5/1095 Loss: 0.4469234049320221
```

Notre TEC s'est amélioré de 1% environ tandis que la précision des mots s'est amélioré de 2%.

Voila le résultat de la 51 époque :

```
Character error rate: 10.589543937708564%. Word accuracy: 73.93340270551508%.
Character error rate improved, save model
Epoch: 51
Train NN
Epoch: 51 Batch: 1/1095 Loss: 0.7415744066238403
Epoch: 51 Batch: 2/1095 Loss: 0.5376996397972107
Epoch: 51 Batch: 3/1095 Loss: 0.5451657772064209
Epoch: 51 Batch: 4/1095 Loss: 0.5271891951560974
Epoch: 51 Batch: 5/1095 Loss: 0.5066370964050293
Epoch: 51 Batch: 6/1095 Loss: 0.6099264025688171
Epoch: 51 Batch: 7/1095 Loss: 1.332318902015686
```

Le TEC est de 10.58% tandis que la précision des mots est proche des 74%

c'est la dernière époque où il y a eu une amélioration et dorénavant toutes les époques qui suivront ne verront pas d'amélioration est c'est ce que nous allons voir.

Epoque 55 :

```
Character error rate: 10.901001112347053%. Word accuracy: 73.15296566077004%.
Character error rate not improved, best so far: 10.901001112347053%
Epoch: 55
Train NN
Epoch: 55 Batch: 1/1095 Loss: 0.4422900676727295
Epoch: 55 Batch: 2/1095 Loss: 0.7587248086929321
Epoch: 55 Batch: 3/1095 Loss: 0.5074245929718018
```

Epoque 61 :

```
Character error rate: 10.852057842046719%. Word accuracy: 73.49982656954562%.
Character error rate not improved, best so far: 10.852057842046719%
Epoch: 61
Train NN
Epoch: 61 Batch: 1/1095 Loss: 0.4643314480781555
Epoch: 61 Batch: 2/1095 Loss: 0.8464576601982117
Epoch: 61 Batch: 3/1095 Loss: 0.7186367511749268
```

Epoque 66 :

```
Character error rate: 10.740823136818687%. Word accuracy: 73.88137356919874%.
Character error rate not improved, best so far: 10.740823136818687%
Epoch: 66
Train NN
Epoch: 66 Batch: 1/1095 Loss: 0.70139479637146
Epoch: 66 Batch: 2/1095 Loss: 0.29578495025634766
Epoch: 66 Batch: 3/1095 Loss: 0.34823837876319885
Epoch: 66 Batch: 4/1095 Loss: 0.7920174598693848
Epoch: 66 Batch: 5/1095 Loss: 0.7021568417549133
```

Epoque 70 :

```
Character error rate: 10.896551724137932%. Word accuracy: 73.60388484217829%.
Character error rate not improved, best so far: 10.896551724137932%
Epoch: 70
Train NN
Epoch: 70 Batch: 1/1095 Loss: 0.48944205045700073
Epoch: 70 Batch: 2/1095 Loss: 0.8250146508216858
Epoch: 70 Batch: 3/1095 Loss: 0.38992583751678467
Epoch: 70 Batch: 4/1095 Loss: 0.22934366762638092
Epoch: 70 Batch: 5/1095 Loss: 0.314309686422348
Epoch: 70 Batch: 6/1095 Loss: 0.30300915241241455
```

Epoque 75 :

```
Character error rate: 10.771968854282536%. Word accuracy: 73.6732570239334%.
Character error rate not improved, best so far: 10.771968854282536%
Epoch: 75
Train NN
Epoch: 75 Batch: 1/1095 Loss: 0.1757805198431015
Epoch: 75 Batch: 2/1095 Loss: 0.6170632839202881
Epoch: 75 Batch: 3/1095 Loss: 0.5805587768554688
Epoch: 75 Batch: 4/1095 Loss: 0.3711228668689728
```

Notre réseau de neurones ne présente plus de meilleurs résultats depuis 25 époques, et notre apprentissage s'arrête maintenant puisqu'il est programmé pour s'arrêter au bout de 25 époques sans amélioration.

```
Character error rate: 10.985539488320356%. Word accuracy: 73.53451266042316%.
Character error rate not improved, best so far: 10.985539488320356%
No more improvement since 25 epochs. Training stopped.
```

Maintenant que notre réseau est entrainé, nous allons présenter nos deux algorithmes utilisés pour la détection et la reconnaissance qui contient aussi les commandes et instructions dédiées au lancement de notre apprentissage ci-dessus.

Il faut noter que notre machine a mis 20 heures pour terminer l'apprentissage, alors que notre deuxième machine aurait pris plusieurs jours.

III.3.1 Interprétation des résultats et statistiques :

-Les pertes a travers les époques.

-Le taux d'érreur de caractère a travers les époques.

-La précision des mots a travers les époques.

Voilà ci-dessous les graphes qui représentent leur évolution :

Les pertes :

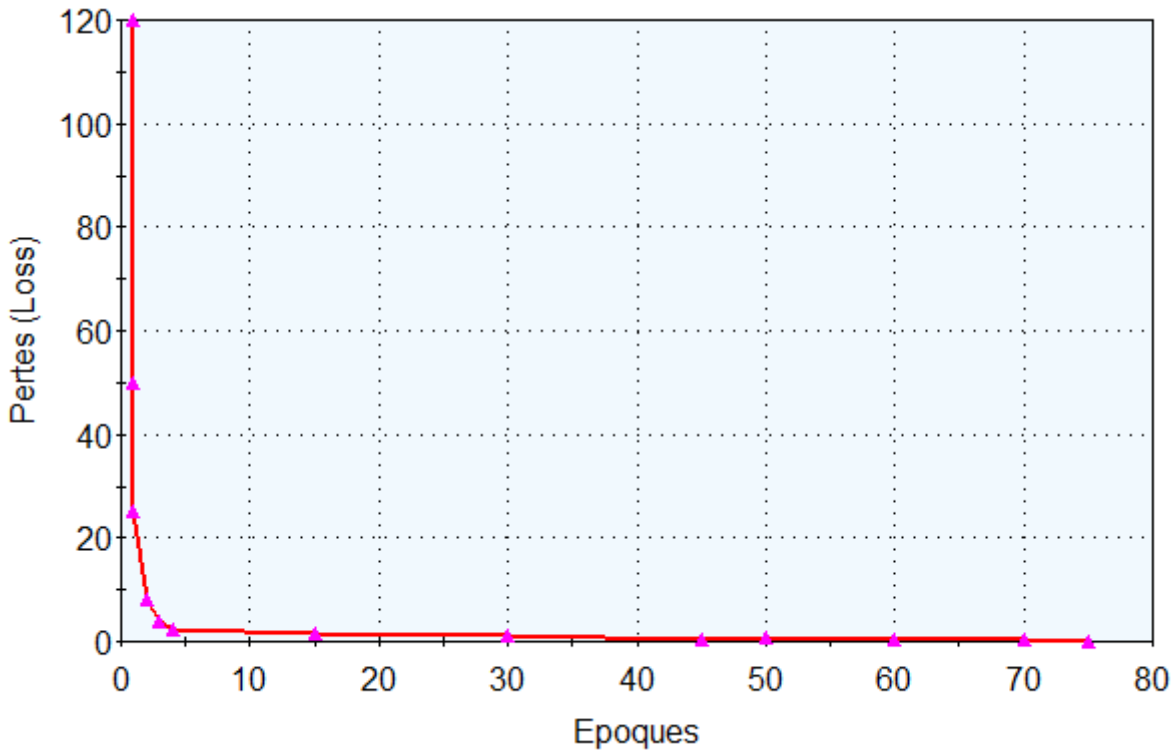


Figure 24 : evolution des pertes par rapport au époques

On constate que plus de 90% des pertes se fait dans les premières époques et plus précisément la première époque.

Après la première époque l'amélioration des pertes ralenti considérablement et on conclue que cette dernière est de loin la plus importante.

Le TEC :

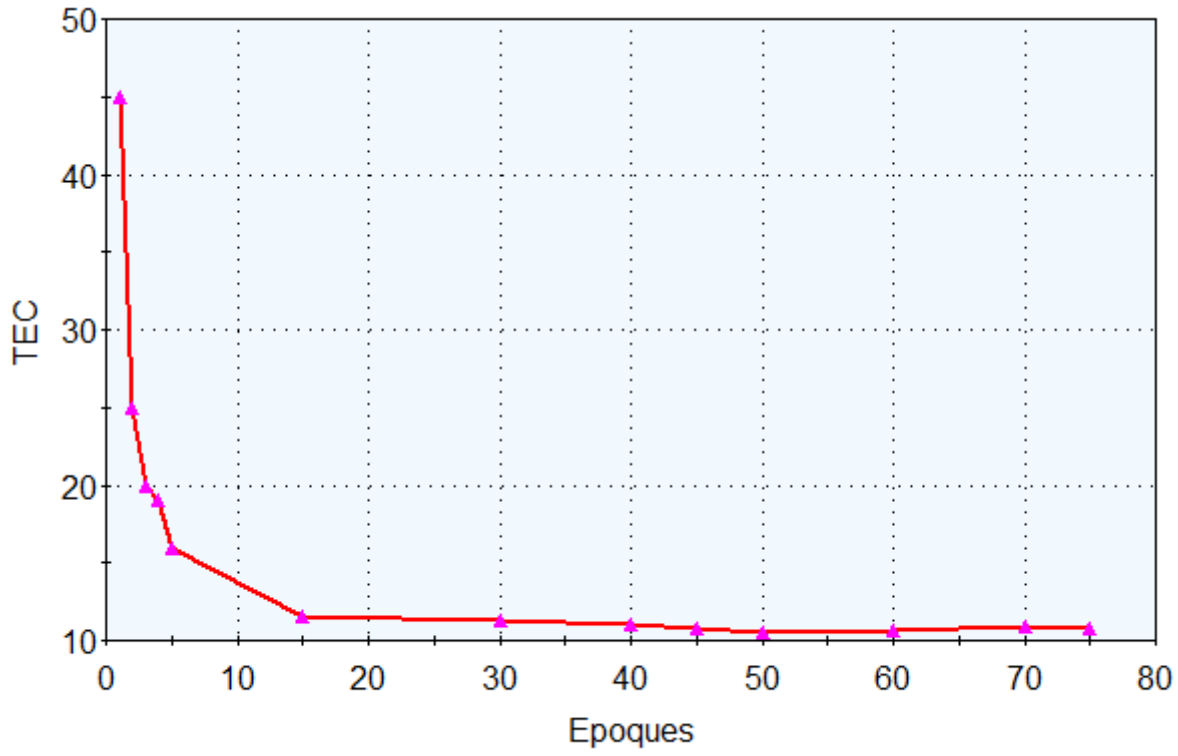


Figure 25 : : graphe qui montre le progrès du TEC

On constate toujours que le taux d'erreur de caractère s'améliore considérablement dans les premières époques et plus précisément dans notre première époque.

Après cela on le taux s'arrête de s'améliorer beaucoup au bout de la 15éme époque et stagne jusqu'à la 75éme époque.

La précision des mots :

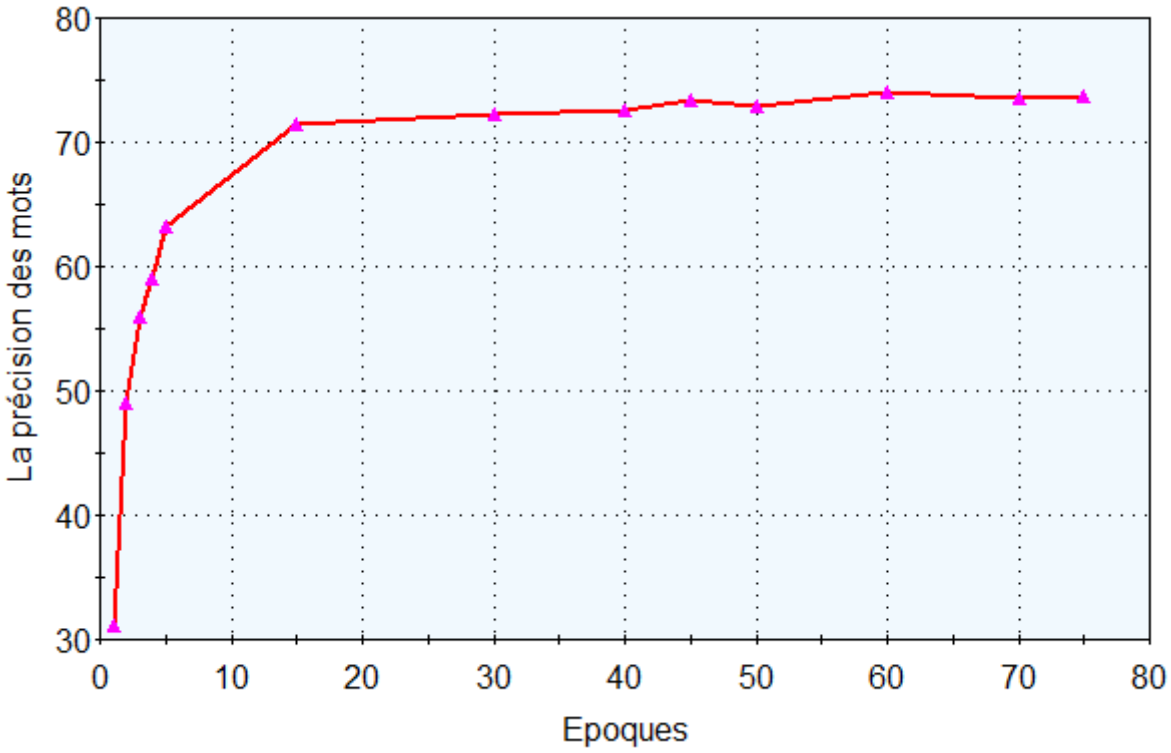


Figure 26 : graphe qui montre le progrès de la précision des mots

Pour la précision des mots c'est la même chose sauf qu'à la place que la précision diminue, elle augmente ce qui est tout a fait logique car notre algorithme apprend d'époque a époque a mieux reconnaitre les mots dans les images de validation.

Le progrès se fait dans les cinq premières époques jusqu'à ce qu'il stagne au bout de la 15éme époque.

III.4 Algorithmes utilisés :

Car notre algorithme ne pouvait reconnaitre que des mots isolés a la fois ou une petite combinaison de mots au plus, alors que nous cherchons a reconnaitre un paragraphe ou texte, nous avons implémenter un deuxième algorithme qui servira a détecter les mots dans une image donnée en entrée.

Cet algorithme servira a identifier et séparer les espaces entre les mots et en ensuite sauvegarder chaque mot en une image séparé dans un dossier a part dans notre ordinateur. On va spécifier le chemin de ce dernier dans notre algorithme de reconnaissance afin qu'il parviendra a les traiter un par un et la sortie de cet algorithme sera une suite de mots traités qui formeront un texte a la fin.

III.4.1 Algorithme de reconnaissance de caractère manuscrits :

Ce sera l'algorithme responsable a l'exécution de notre apprentissage ainsi que faire la reconnaissance de texte manuscrit.

III.4.1.1 Présentation du code :

On commence par ajouter les bibliothèques nécessaires pour notre travail :

```python
import argparse
import json
from typing import Tuple, List
import os
import cv2
import editdistance
from path import Path

from dataloader_iam import DataLoaderIAM, Batch
from model import Model, DecoderType
from preprocessor import Preprocessor

```

Ensuite on crée une class nommé FilePaths et on donne tout les chemins du

```python
class FilePaths:

    fn_char_list = r'C:\Users\razmine\Desktop\memoire\SimpleHTR-master\model\charList.txt'
    fn_summary = r'C:\Users\razmine\Desktop\memoire\SimpleHTR-master\model\summary.json'
    fn_corpus = r'C:\Users\razmine\Desktop\memoire\SimpleHTR-master\data\corpus.txt'


result = []

```

On fixe la hauteur de notre réseau de neuronnes.

```python
def get_img_height() -> int:

    return 32


```

Ici La hauteur est fixée pour notre réseau de neurones, la largeur est définie en fonction du mode d'apprentissage (mots simples ou lignes de texte).

```python
def get_img_size(line_mode: bool = False) -> Tuple[int, int]:

    if line_mode:
        return 256, get_img_height()
    return 128, get_img_height()

```

Ceci Rédige un fichier récapitulatif d'entraînement pour notre réseau de neurones

```python
def write_summary(char_error_rates: List[float], word_accuracies: List[float]) -> None:

    with open(FilePaths.fn_summary, 'w') as f:
        json.dump({'charErrorRates': char_error_rates, 'wordAccuracies': word_accuracies}, f)


def char_list_from_file() -> List[str]:
    with open(FilePaths.fn_char_list) as f:
        return list(f.read())
```

Dans cette section, nous avons le code qui entrainerait l'ensemble de notre dataset si nous avions une meilleure configuration de la machine.

On commence par spécifier le nombre d'itération dès le début qui est 0ensuite Nous implémentons une boucle *while* qui augmente les epochs et maintiendrait l'entrainement tant que le meilleur taux d'erreur de caractère de validation n'est pas satisfait.

```python
def train(model: Model,
          loader: DataLoaderIAM,
          line_mode: bool,
          early_stopping: int = 25) -> None:

    epoch = 0  # number of training epochs since start
    summary_char_error_rates = []
    summary_word_accuracies = []
    preprocessor = Preprocessor(get_img_size(line_mode), data_augmentation=True, line_mode=line_mode)
    best_char_error_rate = float('inf')  # best validation character error rate
    no_improvement_since = 0  # number of epochs no improvement of character error rate occurred
    # stop training after this number of epochs without improvement
    while True:
        epoch += 1
        print('Epoch:', epoch)

        # train
        print('Train NN')
        loader.train_set()
```

On implémente une condition *If* qui enregistre notre model si :

Meilleur taux d'erreur de caractère de validation est supérieur > Taux d'erreur de caractère

Sinon on continue l'entrainement jusqu'à ce que Taux d'erreur de caractère s'améliore.

```
        while loader.has_next():
            iter_info = loader.get_iterator_info()
            batch = loader.get_next()
            batch = preprocessor.process_batch(batch)
            loss = model.train_batch(batch)
            print(f'Epoch: {epoch} Batch: {iter_info[0]}/{iter_info[1]} Loss: {loss}')

        # validate
        char_error_rate, word_accuracy = validate(model, loader, line_mode)

        # write summary
        summary_char_error_rates.append(char_error_rate)
        summary_word_accuracies.append(word_accuracy)
        write_summary(summary_char_error_rates, summary_word_accuracies)

        # if best validation accuracy so far, save model parameters
        if char_error_rate < best_char_error_rate:
            print('Character error rate improved, save model')
            best_char_error_rate = char_error_rate
            no_improvement_since = 0
            model.save()
        else:
            print(f'Character error rate not improved, best so far: {char_error_rate * 100.0}%')
            no_improvement_since += 1
```

Si le Taux d'erreur de caractère s'arréte de s'améliorer donc on termine l'entrainement.

```
        if no_improvement_since >= early_stopping:
            print(f'No more improvement since {early_stopping} epochs. Training stopped.')
            break

```

On prépare la validation de notre réseau de neurones.

```
def validate(model: Model, loader: DataLoaderIAM, line_mode: bool) -> Tuple[float, float]:

    print('Validate NN')
    loader.validation_set()
    preprocessor = Preprocessor(get_img_size(line_mode), line_mode=line_mode)
    num_char_err = 0
    num_char_total = 0
    num_word_ok = 0
    num_word_total = 0
    while loader.has_next():
        iter_info = loader.get_iterator_info()
        print(f'Batch: {iter_info[0]} / {iter_info[1]}')
        batch = loader.get_next()
        batch = preprocessor.process_batch(batch)
        recognized, _ = model.infer_batch(batch)
```

On spécifie deux statistiques a partir de cette section :

- Le taux d'erreur de caractère
- La précision des mots

  Le Tec (Taux d'erreur de caractère) se calcule comme suit :

Tec = nombre de caractère erronés/ nombre de caractére totale.

Pour obtenir son pourcentage on la multiplie par 100

Tandis que la précision des mots se calcule comme suit :

Précision des mots = nombre de mots justes / nombre de mots total

On multiplie aussi par 100 pour obtenir son pourcentage

```
113
114         print('Ground truth -> Recognized')
115         for i in range(len(recognized)):
116             num_word_ok += 1 if batch.gt_texts[i] == recognized[i] else 0
117             num_word_total += 1
118             dist = editdistance.eval(recognized[i], batch.gt_texts[i])
119             num_char_err += dist
120             num_char_total += len(batch.gt_texts[i])
121             print('[OK]' if dist == 0 else '[ERR:%d]' % dist, '"' + batch.gt_texts[i] + '"', '->',
122                   '"' + recognized[i] + '"')
123
124     # print validation result
125     char_error_rate = num_char_err / num_char_total
126     word_accuracy = num_word_ok / num_word_total
127     print(f'Character error rate: {char_error_rate * 100.0}%. Word accuracy: {word_accuracy * 100.0}%.')
128     return char_error_rate, word_accuracy
129
```

Cet algorithme aidera à reconnaître un texte dans une image fournie par le chemin du fichier de cette image spécifique

On note qu'on utilisera la bibliothèque Open-cv afin de réaliser cette tache.

```
130
131 def infer(model: Model, fn_img: Path) -> None:
132     _somme = 0
133     length = len(os.listdir(fn_img))
134     for filename in os.listdir(fn_img):
135         f = os.path.join(fn_img, filename)
136
137         if os.path.isfile(f):
138             img = cv2.imread(f, cv2.IMREAD_GRAYSCALE)
139             assert img is not None
140             preprocessor = Preprocessor(get_img_size(), dynamic_width=True, padding=16)
141             img = preprocessor.process_img(img)
142             batch = Batch([img], None, 1)
143             recognized, probability = model.infer_batch(batch, True)
144             _somme = _somme + probability[0]
145             result.append(recognized[0])
146     print(_somme / length)
147     print(result)
148
149
```

Analyse les arguments de la ligne de commande .

Nous allons ajouter des arguments qui vont nous aider à naviguer et choisir ce que nous voulons faire pendant notre entrainement et préciser le mode de d'entrainement.

Comme exemple on prend les arguments suivants :

--mode : cet argument nous donnera le choix entre entrainer, valider et déduction.

--decoder : cet argument nous permettra de choisir entre les méthodes de décodage (bestpath,beamsearch et wordbeamsearch)

--batch_size : cet argument spécifie la taille du lot, la taille défaut du lot est de 100.

--data_dir : cet argument donne le chemin du dossier contenant la base de donnée IAM.

--fast : c'est argument qui accélére le chargement d'échantillons

--img_file :spécifie le chemin de l'image utilisée comme cible

```
151  def parse_args() -> argparse.Namespace:
152      parser = argparse.ArgumentParser()
153
154      parser.add_argument('--mode', choices=['train', 'validate', 'infer'], default='infer')
155      parser.add_argument('--decoder', choices=['bestpath', 'beamsearch', 'wordbeamsearch'], default='bestpath')
156      parser.add_argument('--batch_size', help='Batch size.', type=int, default=100)
157      parser.add_argument('--data_dir', help='Directory containing IAM dataset.', type=Path, required=False)
158      parser.add_argument('--fast', help='Load samples from LMDB.', action='store_true')
159      parser.add_argument('--line_mode', help='Train to read text lines instead of single words.', action='store_true'
160      parser.add_argument('--img_file', help='Image used for inference.', type=Path, default='../data/word.png')
161      parser.add_argument('--early_stopping', help='Early stopping epochs.', type=int, default=25)
162      parser.add_argument('--dump', help='Dump output of NN to CSV file(s).', action='store_true')
163
164      return parser.parse_args()
165
166
```

Ici on définit notre fonction mais qui est le point d'éxécution de notre programme.

On définit aussi les types de décodeurs qu'on pourra utiliser qui sont :

-Bestpath

-Beamsearch

-Wordbeamsearch

Wordbeamsearch est le Meilleur décodeur d'entre eux et on précisera pourquoi plus tard.

```python
def main():

    # parse arguments and set CTC decoder
    args = parse_args()
    decoder_mapping = {'bestpath': DecoderType.BestPath,
                       'beamsearch': DecoderType.BeamSearch,
                       'wordbeamsearch': DecoderType.WordBeamSearch}
    decoder_type = decoder_mapping[args.decoder]

    # train the model
    if args.mode == 'train':
        loader = DataLoaderIAM(args.data_dir, args.batch_size, fast=args.fast)

        # when in line mode, take care to have a whitespace in the char list
        char_list = loader.char_list
        if args.line_mode and ' ' not in char_list:
            char_list = [' '] + char_list

        # save characters and words
        with open(FilePaths.fn_char_list, 'w') as f:
            f.write(''.join(char_list))
```

```python
        with open(FilePaths.fn_corpus, 'w') as f:
            f.write(' '.join(loader.train_words + loader.validation_words))

        model = Model(char_list, decoder_type)
        train(model, loader, line_mode=args.line_mode, early_stopping=args.early_stopping)

    # evaluate it on the validation set
    elif args.mode == 'validate':
        loader = DataLoaderIAM(args.data_dir, args.batch_size, fast=args.fast)
        model = Model(char_list_from_file(), decoder_type, must_restore=True)
        validate(model, loader, args.line_mode)

    # infer text on test image
    elif args.mode == 'infer':
        model = Model(char_list_from_file(), decoder_type, must_restore=True, dump=args.dump)
        infer(model, args.img_file)


if __name__ == '__main__':
    main()
```

Comme nous avons dit précédemment, cet algorithme ne peut traiter que des images contenant au plus une combinaison simple de mots (une ligne maximum)

C'est pour cela qu'on a mis en œuvre ce deuxième algorithme de détection de mots

III.4.1.2 Algorithme de détection de mots :

Cet algorithme de segmentation utilise la méthode « space scale technique » ,Il prend en entrée une image contenant des mots et nous donne les mots détectés comme sortie. En

option, les mots sont triés selon l'ordre de lecture, de haut en bas, de gauche à droite et ensuite enregistré dans un dossier a part.

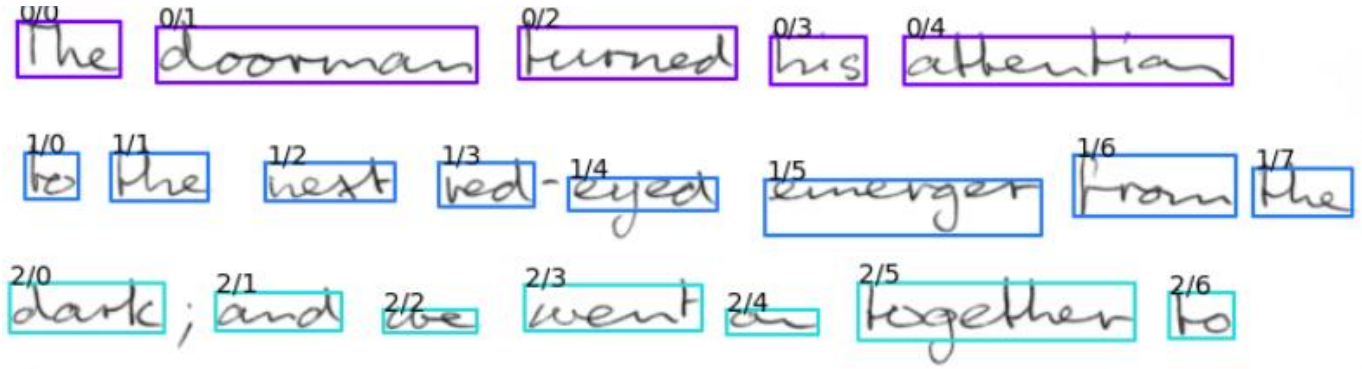


Figure 27 : segmentation des mots a l'aide de l'algorithme de détection

On importe les bibliothèques nécessaires pour notre algorithme

```python
import argparse
from typing import List
import os
import cv2
import matplotlib.pyplot as plt
from path import Path

from word_detector import detect, prepare_img, sort_multiline, sort_line
```

Ceci sert a Renvoyer tous les fichiers image contenus dans un dossier

```python
def get_img_files(data_dir: Path) -> List[Path]:
    res = []
    for ext in ['*.png', '*.jpg', '*.bmp']:
        res += Path(data_dir).files(ext)
    return res
```

Nous ajoutons des arguments pour préciser l'action que nous voulons faire a notre programme de détection de mots.

```python
def main():
    parser = argparse.ArgumentParser()
    parser.add_argument('--data', type=Path, default=Path('../data/line'))
    parser.add_argument('--kernel_size', type=int, default=25)
    parser.add_argument('--sigma', type=float, default=11)
    parser.add_argument('--theta', type=float, default=7)
    parser.add_argument('--min_area', type=int, default=100)
    parser.add_argument('--img_height', type=int, default=50)
    parsed = parser.parse_args()
```

On charge l'image en question et on tri les lignes grâce aux bibliothèques ajoutées.

```
28
29      for fn_img in get_img_files(parsed.data):
30          print(f'Processing file {fn_img}')
31
32          # load image and process it
33          img = prepare_img(cv2.imread(fn_img), parsed.img_height)
34          detections = detect(img,
35                              kernel_size=parsed.kernel_size,
36                              sigma=parsed.sigma,
37                              theta=parsed.theta,
38                              min_area=parsed.min_area)
39
40          # sort detections: cluster into lines, then sort each line
41          lines = sort_multiline(detections)
```

Enfin on dessine et on affiche les résultats après séparation de chaque mot en une image spécifique, qui seront ensuite sauvegardée dans le fichier a part « testData ».

```
43          # plot results
44          num_colors = 7
45          colors = plt.cm.get_cmap('rainbow', num_colors)
46          for line_idx, line in enumerate(lines):
47              for word_idx, det in enumerate(line):
48                  plt.imshow(det.img, cmap='gray')
49                  plt.axis('off')
50                  plt.savefig(f'./testData/x={line_idx}+{word_idx}.png', bbox_inches='tight')
51
52
53  if __name__ == '__main__':
54      main()
55
```

III.5 Wordbeamsearch decoding:

En utilisant notre programme sans le décodeur wordbeamsearch, nous auront des prédictions mauvaises et inexactes, mais en l'incluant on obtiendrait de bien meilleurs résultats.

La recherche par faisceau de mots (word beam search) est un algorithme de décodage CTC. Il est utilisé pour les tâches de reconnaissance de séquences telles que la reconnaissance de texte manuscrit (figure 28) ou la reconnaissance vocale automatique.[37]

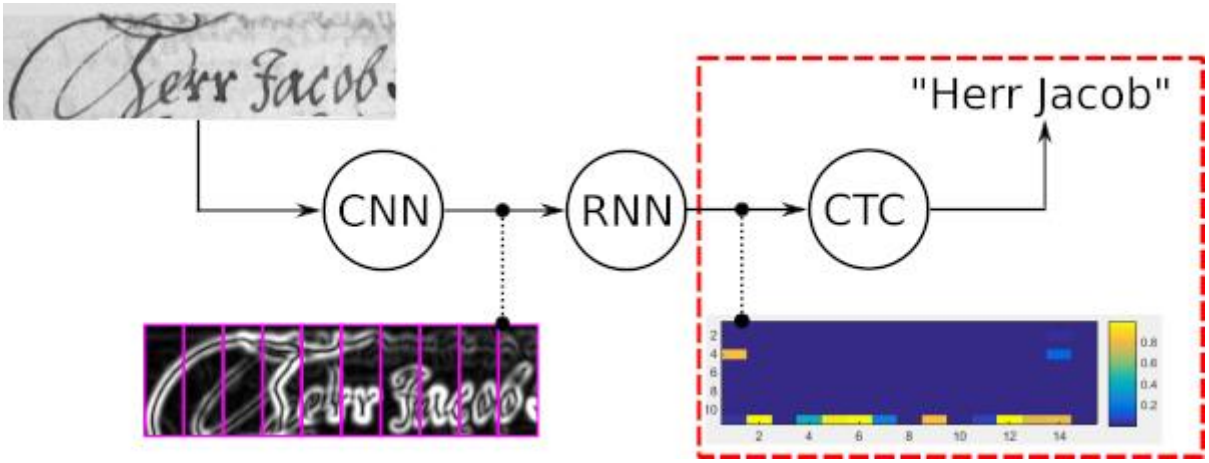


Figure 28 : résultat du traitement après l'application de l'algortithme de wordbeamsearch decoding [37]

III.6 Résultats et application :

Tout d'abord nous allons préciser le chemin de notre image contenant le texte qu'on souhaite traiter.

Elle sera toujours dans le répertoire suivant :
C:\Users\razmine\Desktop\memoire\WordDetector-master\data\page

Voici l'image en question :

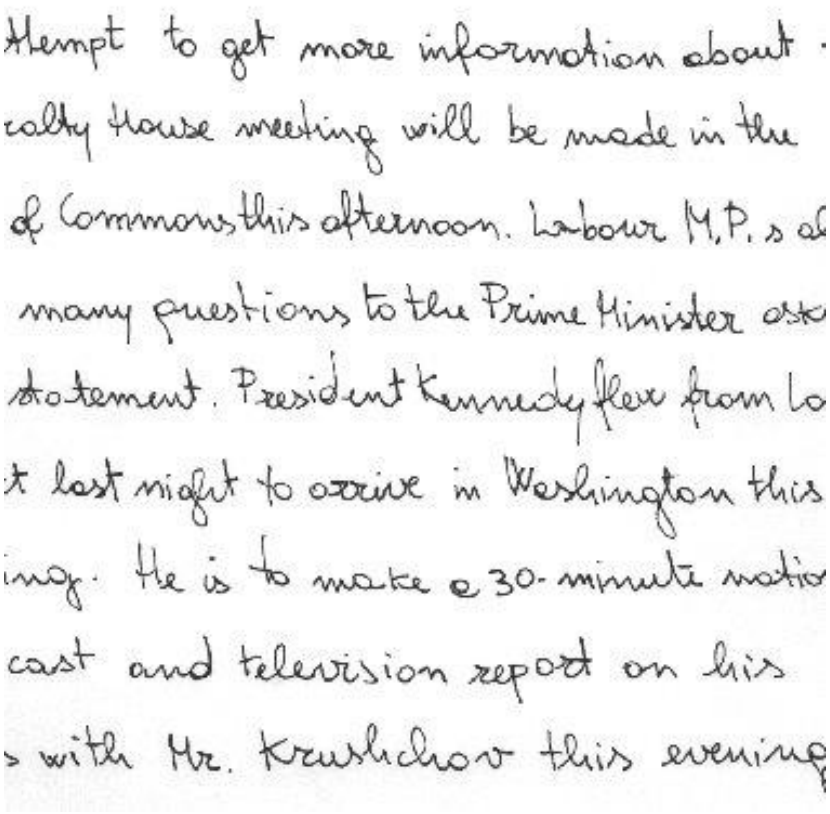


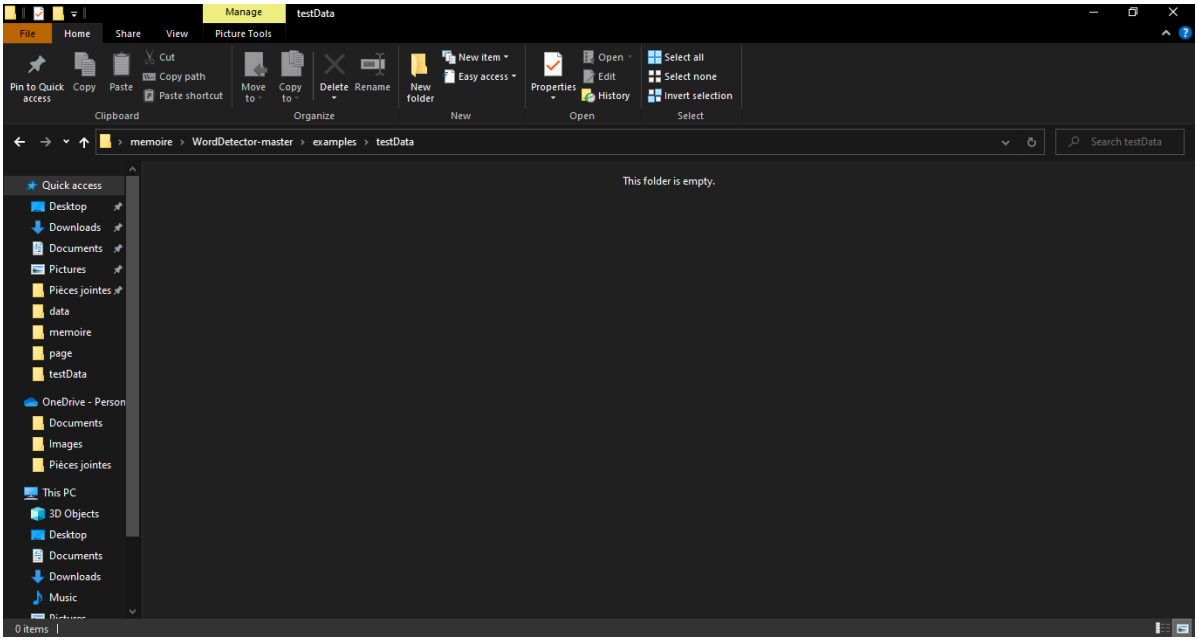
Figure 29 :Image test pour notre algorithme

Tout d'abord nous allons exécuter l'algorithme de détection de mots afin qu'il puisse diviser tous les mots en images distinctes qui seront ensuite enregistrées dans le répertoire spécifié.

Le répertoire en question est :

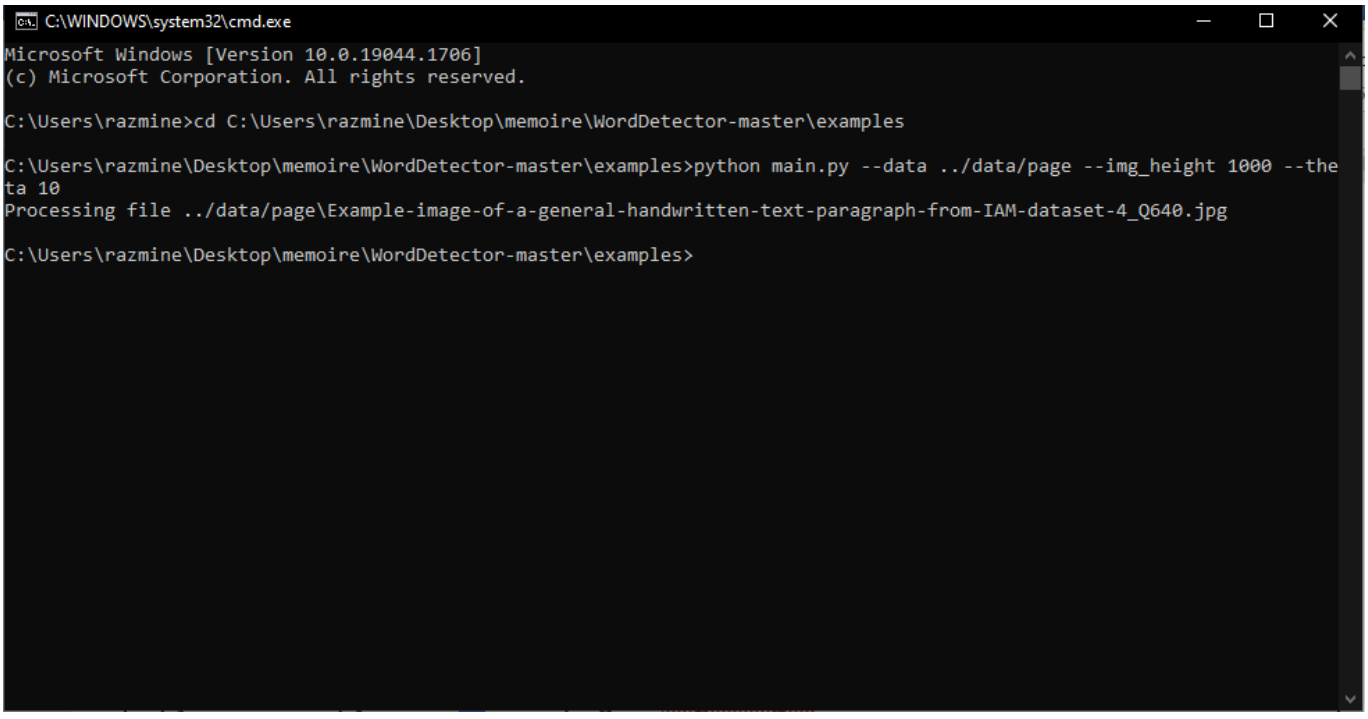
C:\Users\razmine\Desktop\memoire\WordDetector-master\examples\testData

Voila a quoi ressemble notre répertoire avant l'éxécution :

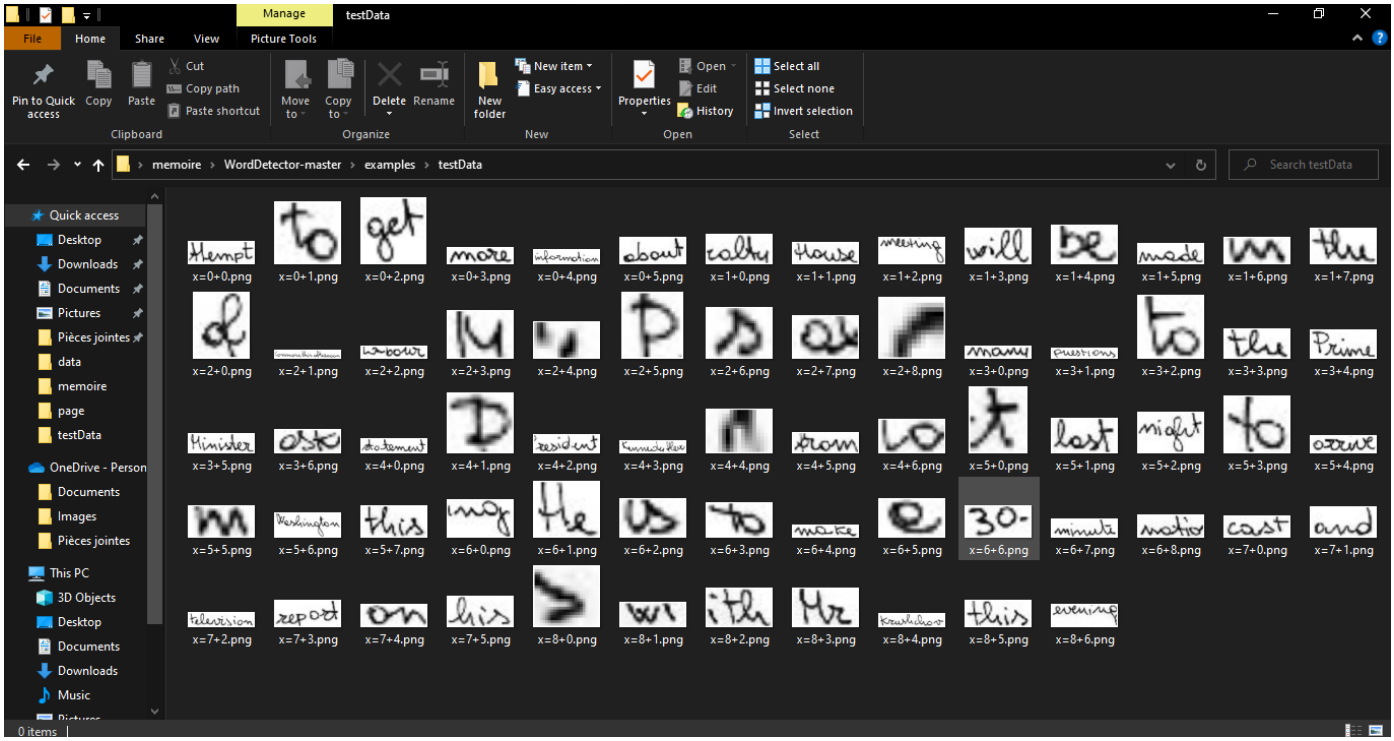


Pour exécuter notre algorithme on lance la commande suivantes avec ses arguments précisés dans l'invite de commandes :

python main.py --data ../data/page --img_height 1000 --theta 10



Voila maintenant notre répertoire après la détection de mots :

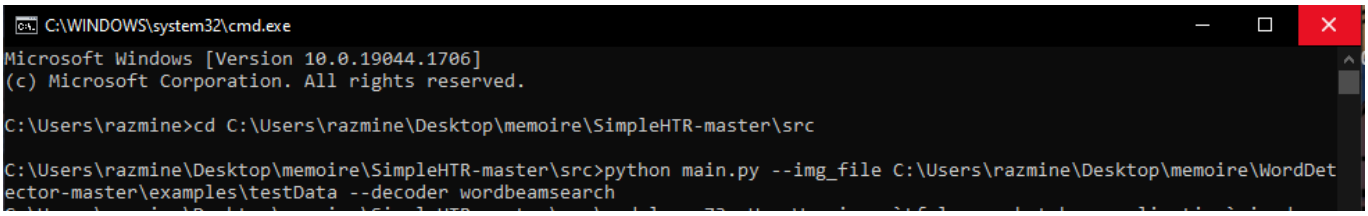


Maintenant que notre texte est segmenté en plusieurs images de mots,

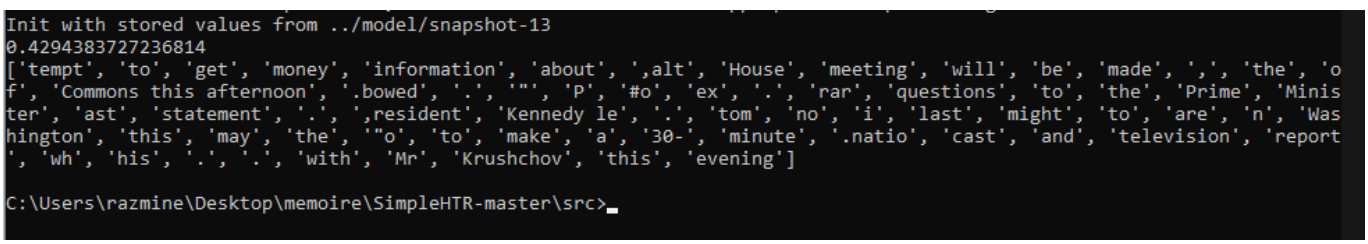
On lance notre algorithme de reconnaissance d'écriture manuscrite .

Pour cela on écrit la commande suivante sur l'invite de commandes :

python main.py --img_file C:\Users\razmine\Desktop\memoire\WordDetector-master\examples\testData --decoder wordbeamsearch



voila le résultat :



Nous avons les mots prédits entre guillemets, et séparés par des virgules

Nous avons aussi calculé la probabilité totale qui est la somme de probabilité de chaque mot divisé par le nombre totale de mots qui est dans notre cas égale a 42.94%.

Deuxième exemple :

Pour notre deuxième exemple nous allons prendre une différente imagine contenante un différent texte qui est illustré ci-dessous :

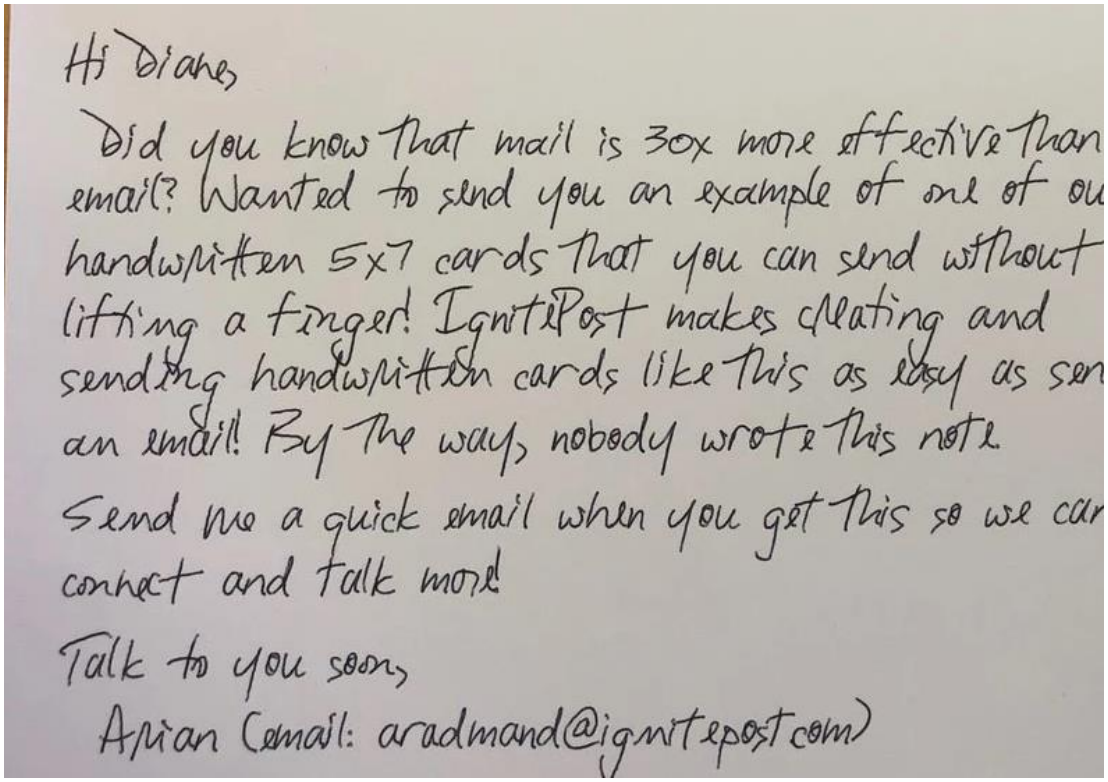
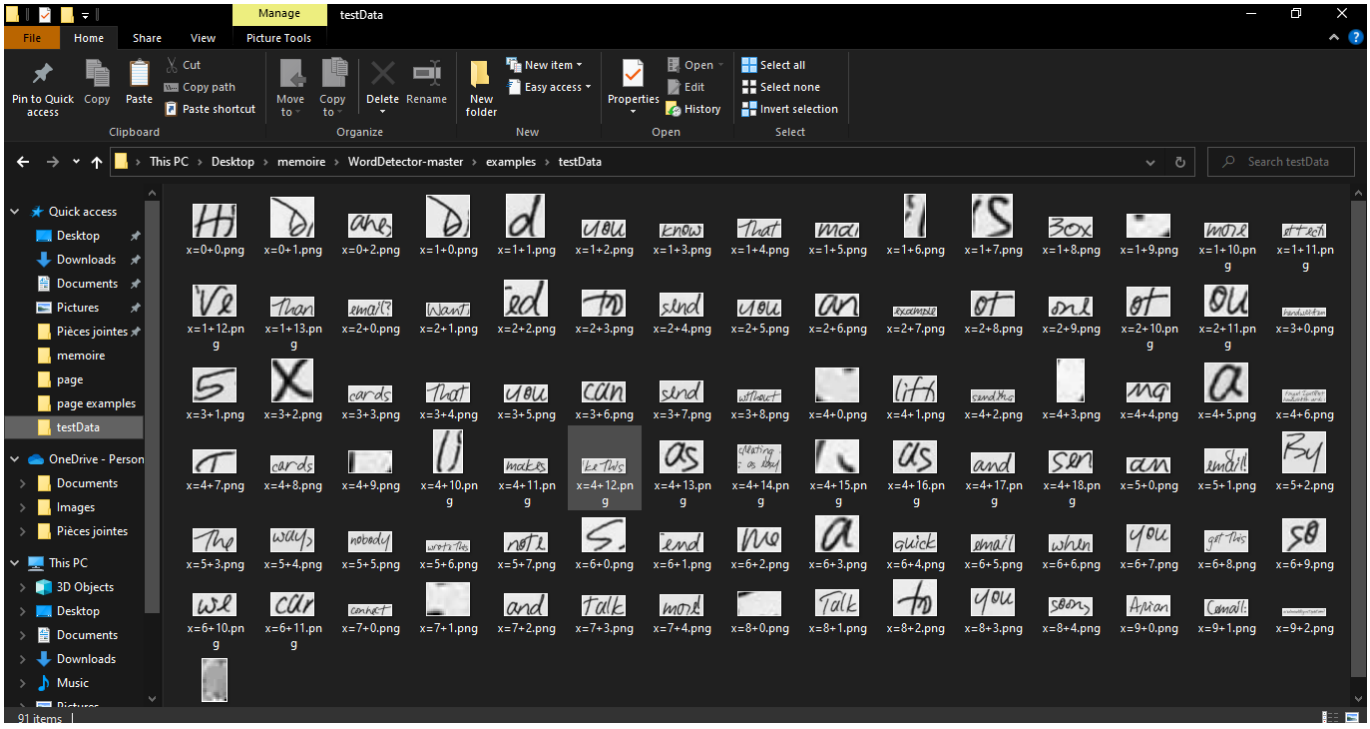
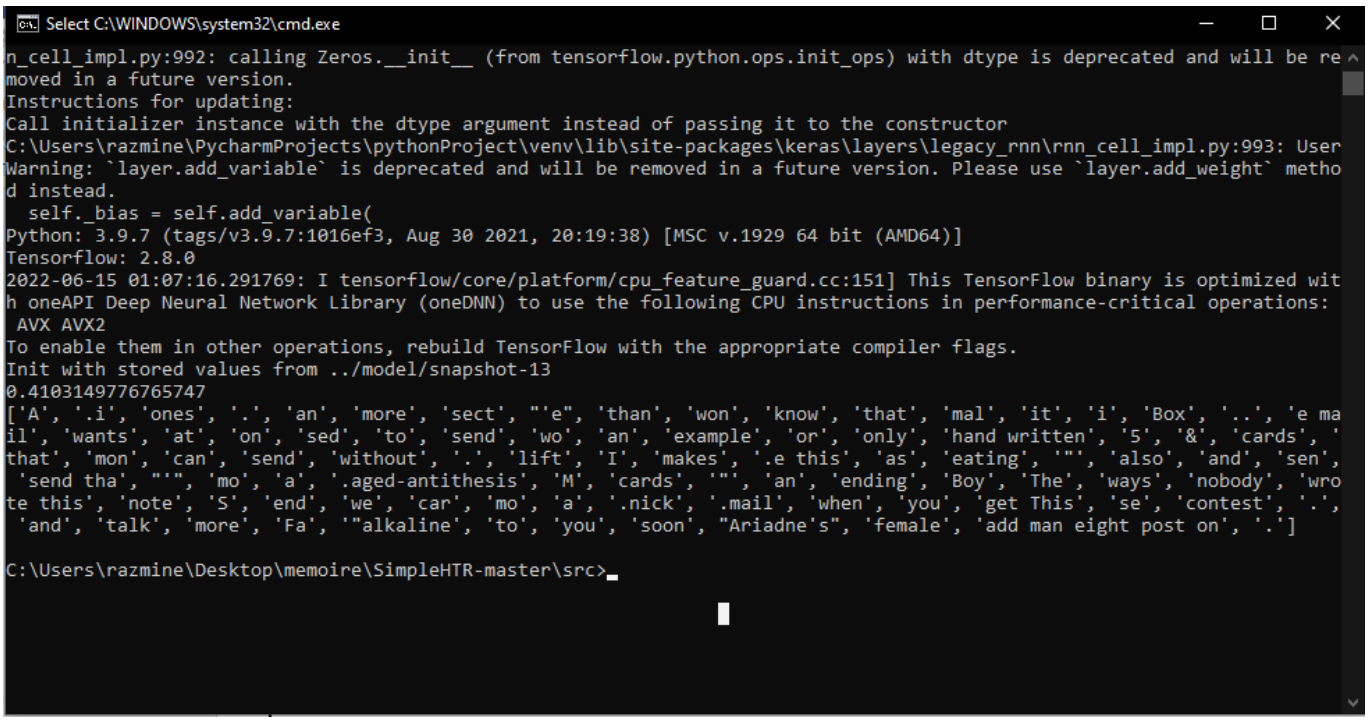


Figure 30 : Deuxiéme exemple pour notre alogithme

On exécute notre premier algorithme pour détecter et enregistrer les mots :



Maintenant que les mots sont séparés on exécute notre algorithme de reconnaissance, voila le résultat :



Notre probabilité totale cette fois-ci est de 41.03%

III.7 Discussion :

On remarque que notre algorithme commet de nombreuses fautes et se trompe souvent, cela est principalement dû aux nombreux types d'écriture et polices que les gens possèdent, donc concevoir un algorithme qui reconnaîtrait nos textes sans erreur

nécessiterait que le dernier soit formé sur une immense base de données qui contient beaucoup de styles et de polices d'écriture manuscrite que celle utilisée pour notre projet.

Nous aurions besoin possiblement de fusionner plusieurs bases de données avec différentes polices et styles d'écriture manuscrite et même d'utiliser la technique d'augmentation des données (data augmentation) pour approfondir notre entrainement et obtenir des résultats encore plus précis.

Un autre problème que notre programme a peut-être rencontré est l'espacement entre les mots, lettres et lignes. le petit espace entre les lignes, les mots et les lettres aurait pu faire croire à notre réseau de neurones que ces deux lettres faisaient partie d'un mot entier alors que ce n'était pas le cas.

Même chose pour le petit espacement entre les lignes car cela aurait fait deviner à notre réseau de neurones que deux lignes différentes faisaient partie de la même ligne.

Ce projet est très important dans le domaine de télécommunication car il nous offre la possibilité de communiquer a travers le monde et nous aide à mieux comprendre et numériser des textes en mauvais états ou dont leurs écriture est difficile à lire.

III.8 **Conclusion** :

Dans ce chapitre nous avons démontré étape par étape l'exécution de notre apprentissage et de notre algorithme ainsi que les résultats obtenus, tout en incluant les statistiques concernant la précision et les pertes de notre apprentissage et programme.

Conclusion générale

Conclusion générale :

Nous avons essayé au cours de cette expérience, de réaliser un système de reconnaissance de texte manuscrit, nous avons également mis en évidence l'état de l'art de plusieurs définitions du deep learning et de la base de données.

Nous avons rencontré quelques difficultés avec le matériel et les logiciels utilisés et nous avons trouvé des solutions afin de régler et de continuer notre projet le plus soigneusement possible.

Et nous avons aussi parler des résultats et des caractéristiques de notre modèle ainsi que celles de notre algorithme

Actuellement il est difficile de mettre en point un système de reconnaissance sans faille et sans erreur puis ce que ca demanderait une énorme base de données avec une machine très puissante, chose qui est très dur de se procurer

Références bibliographiques

Références bibliographques :

[1] Guenter Muehlberger , Louise Seaward , Melissa Terras ,…, « Transforming scholarship in the archives through handwritten text recognition » Handwritten text recognition – an overview. Pp2-3. December 2018.

[2] Afafe Lahreche «Deep learning for Arabic letters recognition»,Application, Thése de master, unviversité de Ouargla, Pp 8-9, 2019.

[3] Dalbir, Sanjiv Kumar Singh «Review of Online & Offline Character Recognition »Types of character recognition, vol4 pp1-2, may 2015

[4] Techno-Science.net «https://www.techno-science.net/definition/10800.html»

[5] Hannahs messy handwriting «https://fontsgeek.com/hannahs-messy-handwriting-font»

[6] Echantillon de la base de donnée utilisée IAM

[7] Celia ANÇA, Noria BOUSSAD « Reconnaissance de mots manuscrits en utilisant les réseaux de neurones Application : aux noms d'auteurs arabes» thèse de master, université TIZI-OUZOU,pp8-9,2016.

[8] Mouatsi Houda, Benzeghda Amina « La reconnaissance des caractères Arabes manuscrits par le mécanisme d'Attention » thèse de maste, Université Larbi Ben M'hidi, Pp10-12, 2020.

[9] Tech-target, artificial neuron «https://www.techtarget.com/searchcio/definition/artificial-neuron»

[10] Djamel Belhaouci, Juri'Predis, le Blog «https://www.juripredis.com/fr/blog/id-19-demystifier-le-machine-learning-partie-2-les-reseaux-de-neurones-artificiels».

[11] Ghaith Ahmad Al-Qudah , «A Multi-Layer Perceptron Neural Network Based-Model for Face Detection», these de master, Middle East University for Graduate Studies,pp25-26, may 2009.

[12] Aref Hashemi ,FathFarshid ,MadanifarMasood Abbasi1 «Implementation of multilayer perceptron (MLP) and radial basis function (RBF) neural networks to predict solution gas-oil ratio of crude oil systems» page 2,2020.

[13] Jason Brownlee, «How to Choose an Activation Function for Deep Learning»,Machine learning Mastery «https://machinelearningmastery.com/choose-an-activation-function-for-deep-learning/#:~:text=An%20activation%20function%20in%20a,a%20layer%20of%20the%20network. »january 18 2021.

[14] Sebastian Raschka, Activation Functions for Artificial Neural Networks , «https://www.kaggle.com/getting-started/211865» 2021.

[15] KHERMAZA ELYES, BOUTIARA ABDELOUAHAB «Traitement des images IRM pour la détection des tumeurs cérébrales par les algorithmes de Deep Learning CNN, Faster RCNN, Mask R-CNN et Transfer Learning sous environnement Cloud» thèse de master, université de blida, pp17,2020

[16] Bastien L« https://datascientest.com/machine-learning-tout-savoir» Machine Learning : Définition, fonctionnement, utilisations Novembre 2020

[17] BELHADJER Hakim,SAROUER Brahim «Classification des images avec les réseaux de neurones Convolutionnels» thèse de master, Université de TIZI-OUZOU, pp19

[18] Juri'Predis«https://www.juripredis.com/fr/blog/id-19-demystifier-le-machine-learning-partie-2-les-reseaux-de-neurones-artificiels#:~:text=Le%20neurone%20est%20une%20unit%C3%A9,des%20t%C3%A2ches%20de%20Machine%20Learning.» que sont les réseaux de neurones ?

[19] Actualité informatique« https://actualiteinformatique.fr/intelligence-artificielle/definition-deep-learning» Définition du deep learning, fonctionnement

[20] Yugesh Verma, analytics in diamag, «https://analyticsindiamag.com/r-cnn-vs-fast-r-cnn-vs-faster-r-cnn-a-comparative-guide/» R-CNN vs Fast R-CNN vs Faster R-CNN – A Comparative Guide

[21] Lin Jiang «https://www.hindawi.com/journals/cin/2021/9945934/» Volume 2021, 24 Sep 2021

[22] « https://dataanalyticspost.com/Lexique/reseau-de-neurones-profonds/» RÉSEAU DE NEURONES PROFONDS

[23] Mokri Mohammed Zakaria, «Classification des images avec les réseaux de neuronesconvolutionnels» 2017.

[24] Acervo Lima «https://fr.acervolima.com/introduction-au-reseau-de-neurones-recurrents/» Introduction au réseau de neurones récurrents.

[25] IAM Handwriting Database «https://fki.tic.heia-fr.ch/databases/iam-handwriting-database»

[26] Siddhant's Scratch Book « https://sid2697.github.io/Blog_Sid/algorithm/2019/10/19/CTC-Loss.html» Oct 19, 2019

[27] Harald Sheidl « https://towardsdatascience.com/build-a-handwritten-text-recognition-system-using-tensorflow-2326a3487cd5» Jun 15, 2018

[28] https://www.futura-sciences.com/tech/definitions/informatique-python-19349/

[29] https://fr.wikipedia.org/wiki/PyCharm

[30] https://datascientest.com/tensorflow

[31]https ://datascientest.com/keras

[32] https://www.tutorialspoint.com/how-to-install-opencv-in-python

[33] https://pypi.org/project/editdistance/0.3.1/

[34] https://www.activestate.com/resources/quick-reads/what-is-matplotlib-in-python-how-to-use-it-for-plotting/#:~:text=Matplotlib%20is%20a%20cross%2Dplatform,embed%20plots%20in%20GUI%20applications.

[35] https://www.mygreatlearning.com/blog/python-numpy-tutorial/

[36] https://docs.python.org/3/library/pathlib.html

[37] https://towardsdatascience.com/word-beam-search-a-ctc-decoding-algorithm-b051d28f3d2e

[38] https://fki.tic.heia-fr.ch/databases/iam-handwriting-database site de la base de donnée