REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique
Université Mohamed Khider – BISKRA
Faculté des Sciences Exactes, des Sciences de la Nature et de la Vie
Département d'informatique

N° d'ordre : 201535040396/M2/2020

Mémoire

Présenté pour obtenir le diplôme de master académique en

Informatique

Parcours : **INTELLIGENCE ARTIFICIELLE**

Reconnaissance de caractères de sous-titres vidéo en utilisant l'apprentissage automatique.

Par :

SALMI MOHAMED AMINE BOUZID

Encadrées par :

M.TIBARMACINE AHMED

Remerciements

Avant tout, je remercie Dieu le très haut qui m’aidait et donne-moi le courage et la volonté de réaliser ce modeste travail.
Je remercie le seigneur tout-puissant de m'avoir accordé volonté et patience dans l’accomplissement de ce travail à terme.
Je tiens à exprimer mon vif remerciement M. TIBARMACINE Ahmed qui ma aider malgré la période de la pandémie.
Mes remerciements vont également aux membres du jury d’avoir accepté d’évaluer mon travail.
Sans oublier bien sûr de remercier profondément tous ceux qui ont contribué de près ou de loin à la réalisation de ce travail.

Résumé

Une tâche importante dans l'indexation vidéo basée sur le contenu est d'extraire des informations textuelles des vidéos. Les défis de l'extraction et de la reconnaissance de texte sont la variation de l'éclairage sur chaque cadre vidéo avec du texte, le texte présent sur le fond complexe et la taille de police différente du texte. Nous avons développé un système de reconnaissance de caractères de sous-titre vidéo, spécifiquement adapté pour détecter et reconnaitre les textes incrustés. Reposant sur des approches d'apprentissage profond pour la reconnaissance de caractères. La segmentation, l'extraction des caractéristiques et la classification sont les principales étapes de la reconnaissance des caractères. Plusieurs résultats expérimentaux démontrent la performance des algorithmes proposés.

Mots-clés : Reconnaissance des caractères, Classification, Extractions des caractéristiques, Segmentation, Apprentissage, Réseaux de neurones.

Abstract

An important task in content-based video indexing is to extract textual information from videos. The challenges of text extraction and recognition are the variation of the lighting on each video frame with text, the text present on the complex background and the different font size of the text. We have developed a video subtitle character recognition system, specifically adapted to detect and recognize embedded text. Based on deep learning approaches to character recognition. Segmentation, feature extraction and classification are the main steps in character recognition. Several experimental results demonstrate the performance of the proposed algorithms.

Key-words : Character Recognition, Classification, Feature Extraction, Segmentation, Learning, Neural networks.

Table des matières

Liste des figues

Liste des tableaux

Introduction Générale

Avec la croissance rapide de la base de données multimédia, le contenu de base a suscité l'intérêt de plusieurs experts en traitement d'images à l'ère actuelle. Comme les vidéos portent beaucoup d'informations, parmi lesquelles l'information sémantique est fournie par le texte présent en elle. Dans les résultats expérimentaux obtenus par Judd et al. [1], il a été prouvé que le texte transmet plus d'informations. Le texte présent dans les vidéos est de deux types :

1- texte de scène et **2-** texte graphique.

Le texte ajouté en externe (texte graphique) à la vidéo parlée du contenu de la vidéo. Les sous-titres sont le texte graphique présent dans les vidéos qui aident à l'indexation vidéo basée sur le contenu. Le processus d'analyse des sous-titres comporte cinq étapes : (1) Extraction des images de la vidéo, (2) pré-traitement d'images, (3) Segmentation des caractères présents dans les sous-titres, (4) Extraction de caractéristiques à partir de caractères segmentés et (5) Classification des caractères.

Les principaux défis présents dans l'extraction des sous-titres de la vidéo sont le fond complexe, l'éclairage et la faible résolution. Par conséquent, la segmentation des caractères est mise en œuvre pour réduire la zone non textuelle. Après le processus de segmentation, l'information du personnage peut être connue en extrayant des caractéristiques importantes de celui-ci. Comme les caractéristiques de chaque caractère sont différentes les unes des autres, il peut être utilisé comme paramètre de classification. La classification des caractères est la même que la reconnaissance des caractères. La reconnaissance des caractères des sous-titres aide dans la classification des vidéos telles que les vidéos avec des sous-titres anglais et les vidéos avec d'autres sous-titres de langue.

L'objectif principal de ce travail consiste à concevoir automatiquement un système autonome à base réseau de neurones artificiels. Ce système est capable de reconnaître les caractères de sous-titres vidéo en utilisant les techniques de l'apprentissage profond (Deep learning « DL »).

Le présent mémoire est organisé en trois chapitres dont les thèmes sont donnés ci-dessous : dans un premier temps, nous introduisons la notion des réseaux de neurones artificiels, d'apprentissage automatique et de nouveau paradigme Deep Learning « DL », et dans le deuxième chapitre nous présentons des généralités sur le domaine de reconnaissance des formes

et ses principales caractéristiques et les diffèrent types des formes existantes. Dans le troisième chapitre nous présentons la conception générale de notre système développé de reconnaissance de caractères et l'architecture conceptuelle de deux approches neuronales ainsi les résultats obtenus et une comparaison entre les résultats de deux approches.

Chapitre 01 : Apprentissage machines.

1.1 Introduction

L'apprentissage automatique (en anglais machine learning, littéralement <<l'apprentissage machine>>) ou apprentissage statistique est un champ d'étude de l'intelligence artificielle qui se fonde sur des approches statistiques pour donner aux ordinateurs la capacitée << apprendre >>à partir de données, c'est-à-dire d'améliorer leurs performances à résoudre des tâches sans être explicitement programmés pour chacune. Plus largement, cela concerne la conception, l'analyse, le développement et l'implémentation de telles méthodes.

1.2 Objectif de L'Apprentissage machine

L'apprentissage machine est une sous-discipline de l'intelligence artificiel dont l'objectif ultime est de reproduire chez l'ordinateur les capacités cognitives de l'être humain, par *le biais de l'apprentissage*. Ici, le terme apprentissage est à mettre en opposition avec des techniques d'intelligence artificielle basées sur des comportements intelligents "pré-codés", comme dans le fameux programme ELIZA [9], où l'ordinateur donnait l'illusion de pouvoir poursuivre une conversation intelligente avec un humain à partir d'un système en fait très basique de détection de mots-clés et de réponses prédéfinie. L'approche "apprentissage machine" consiste plutôt à programmer des mécanismes qui permettent de développer les connaissances c'est-à-dire d'apprendre à partir d'observations (*les exemples d'apprentissage*) de manière automatique [26].

Les êtres humains faisant preuve de capacités d'apprentissage impressionnantes, il est naturel que l'apprentissage machine s'inspire d'eux pour tenter de reproduire leurs comportements. En particulier, les travaux en neurosciences visant à comprendre les mécanismes d'apprentissage dans le cerveau. Une classe d'algorithmes d'apprentissage très populaire, les réseaux de neurones, a ainsi été inspirée de telles observations biologiques. Mais les détails du fonctionnement du cerveau restant pour l'instant en grande partie un mystère. Les algorithmes développés en apprentissage machine ont des objectifs moins ambitieux. Ils ont chacun leurs propres forces et faiblesses – souvent très différentes de celles des humains – et sont généralement prévus pour résoudre certains types de problèmes spécifiques [26].

1.3 **Différents types d'apprentissage**

1.3.1 Apprentissage supervisé

La forme d'apprentissage qui est considérée comme la plus intuitive est celle dite d'apprentissage supervisé. Cela signifie que la réponse attendue est observée dans les données collectées. Formellement, si l'on note *z* un exemple d'apprentissage, alors on peut écrire $z = (x, y)$ avec *x* la partie "entrée", c'est-à-dire les données que l'algorithme est autorisé à utiliser pour faire une prédiction, et *y* la partie "étiquette", c'est-à-dire la valeur correcte pour la prédiction. Par exemple, si la tâche consiste à déterminer le sexe d'une personne à partir d'une photo d'identité, *x* serait la photo et *y* soit "homme", soit "femme" (en supposant qu'il s'agisse des deux seules possibilités). Dans un tel cas où les valeurs possibles de l'étiquette *y* ne sont pas interprétables comme des nombres, on parle de tâche de classification. Si par contre la tâche était de prédire l'âge de la personne plutôt que son sexe, *y* serait un nombre et on parlerait d'une tâche de régression. La figure 1.1 illustre ces deux situations [26].

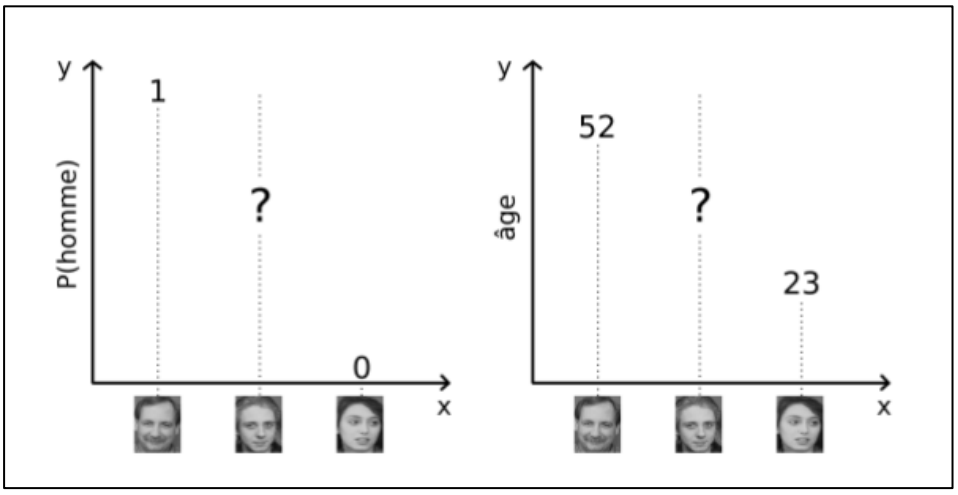


Figure 1. 1: Exemples d'apprentissage supervisé : Classification (à gauche) pour prédire le sexe, et régression (à droite) pour prédire l'âge, Les photos aux deux extrémités de l'axe des x sont des exemples d'entraînement pour lesquels l'étiquette est connue, tandis que la photo du milieu est celle pour laquelle on veut obtenir une prédiction [26].

L'algorithme est *entraîné* sur un ensemble de données pré-collectées (l'ensemble d'entrainement), de la forme $D = \{z_1, \dots, z_n\}$, avec $z_i = (x_i, y_i)$. De nombreux algorithmes supposent que ces exemples sont tirés de manière indépendante et identiquement distribuée d'une distribution P, c'est-à-dire. $z_i \sim P(X, Y)$ (où la forme exacte de P n'est pas connue

d'avance). Selon la tâche à résoudre, la prédiction d'un algorithme d'apprentissage supervisé va généralement tenter d'approximer l'une des trois quantités suivantes [26] :

– $P(x|y)$: dans notre exemple de classification (trouver le sexe), il faudrait prédire la probabilité qu'une photo soit une photo d'un homme par un nombre p (la probabilité que ce soit la photo d'une femme étant alors 1−p). Dans notre exemple de régression (trouver l'âge), la prédiction serait une densité de probabilité sur l'intervalle $]0, +\infty[$.

– $argmax_y P(x|y)$: dans notre exemple de classification, il s'agirait d'une prédiction binaire du sexe le plus probable ("homme" ou "femme"). Dans notre exemple de régression, il s'agirait de l'âge le plus probable.

– $E_Y = [Y|x] = \int_y yP(y|x)\,dy$: cette quantité n'a de sens que pour la régression, et il s'agirait dans notre exemple de prédire l'âge moyen de la personne étant donnée sa photo.

Il faut noter que la prédiction de $P(y|x)$ est la tâche la plus générale (dans la mesure où la résoudre permet également d'accomplir les deux autres), mais tous les algorithmes d'apprentissage ne sont pas capables d'estimer une distribution de probabilité [26].

1.3.2 Apprentissage non supervisé

Dans l'apprentissage non supervisé, un exemple $z_i = x_i \sim P(x)$ ne contient pas d'étiquette explicite. Il existe plusieurs types de tâches d'apprentissage non supervisé, parmi lesquelles les plus souvent rencontrées sont [26] :

- L'estimation de densité : estimer P(x), comme illustré en figure 1.2.
- La génération de données : tirer de nouveaux exemples d'une distribution la plus proche possible de P(X).
- L'extraction de caractéristiques
- Le regroupement (clustering) : partitionner les exemples en groupes $G_1, \ldots, G_k$ tels que tous les exemples dans un même groupe soient similaires

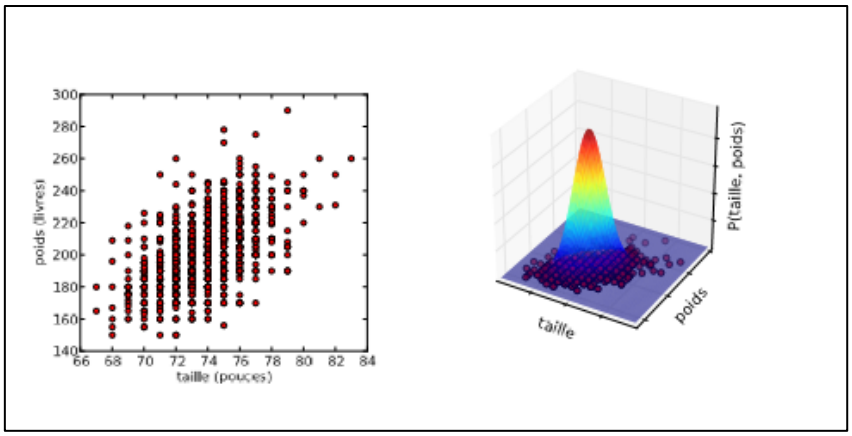


Figure 1. 2 . Exemple d'apprentissage non supervisé : l'estimation de densité. À gauche, les données d'origine (taille et poids de 1033 joueurs de baseball aux Etats-Unis). À droite, l'estimation de la densité par une distribution Gaussienne [26].

1.3.3 Apprentissage semi-supervisé

Comme le nom l'indique, l'apprentissage semi-supervisé est à mi-chemin entre le supervisé et le non supervisé : certains exemples $z_i = (x_i, y_i)(1 \leq i \leq l)$ ont une étiquette, tandis que d'autres exemples $z_i = x_i(l + 1 \leq i \leq n)$ n'en ont pas (on dit qu'ils ne sont pas "étiquetés"). Les algorithmes d'apprentissage semi-supervisé tentent généralement de résoudre des problèmes d'apprentissage supervisé, mais en utilisant les exemples non étiquetés pour améliorer leur prédiction. La distribution des entrées $P(X)$ peut nous donner de l'information sur $P(Y|X)$ même en l'absence d'étiquettes. Un exemple typique où c'est le cas, pour une tâche de classification, est lorsque les exemples de chaque classe forment des groupes distincts dans l'espace des entrées, séparés par des zones de faible densité $P(x)$. La figure 1.3 montre ainsi deux classes dont la forme en croissant est révélée par les exemples non étiquetés. Un

algorithme purement supervisé donc ignorant ces exemples ne pourrait pas idéntifier correctement ces deux classes [26].

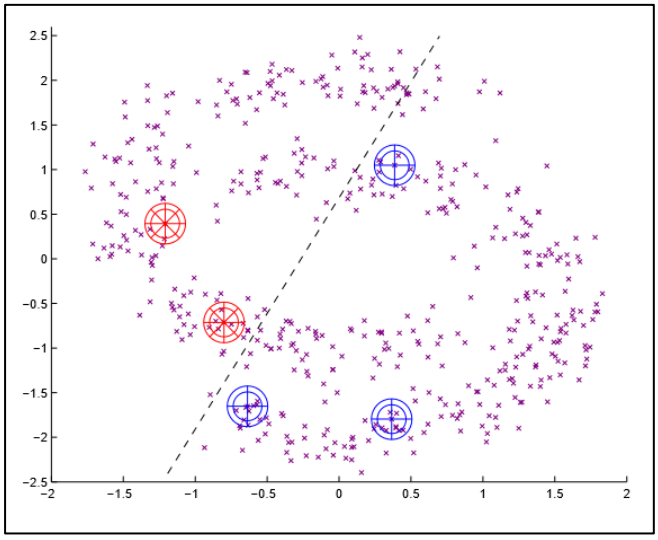


Figure 1. 3 . Apprentissage semi-supervisé : ici seuls 5 exemples sont étiquetés (deux de la classe * en rouge, et trois de la classe + en bleu). Un algorithme n'utilisant pas les exemples non étiquetés (petits x mauves) séparerait les deux classes par exemple selon la ligne en pointillés, alors qu'un algorithme semi-supervisé (ou un humain) pourrait séparer les deux "croissants de lune" correspondant à chaque classe [26].

Dans le cadre de l'apprentissage supervisé décrit précédemment, l'application d'un algorithme se fait typiquement en deux étapes [26] :

1. L'algorithme va d'abord apprendre une tâche (phase d'entraînement) sur un ensemble $D = \{(x_1, y_1), \cdots, (x_n, y_n)\}$.
2. L'algorithme doit ensuite faire ses prédictions (phase de test) sur un ensemble T dans on ne fournit pas les étiquettes.

Un algorithme semi-supervisé peut également suivre ces deux étapes (en n'oubliant pas que D peut contenir également des exemples non étiquetés). On dit alors que la phase de prédiction sur l'ensemble de test se fait par induction. Mais puisque l'algorithme peut utiliser des exemples non étiquetés au cours de l'apprentissage, on peut également inclure T dans D : on parle alors de *transduction*, et en général les performances seront meilleures qu'en induction puisque plus d'exemples sont disponibles pour l'apprentissage. Il existe malgré tous des applications où il n'est pas possible d'inclure T dans l'ensemble d'entraînement, par exemple lorsque les éléments de T sont générés en temps réel et que l'on a besoin de prédictions immédiates : si ré-entraîner l'algorithme avant chaque nouvelle prédiction s'avère trop lent, l'induction est alors la seule option possible [26].

1.3.4 Apprentissage par renforcement

L'apprentissage par renforcement est une forme d'apprentissage supervisé où l'algorithme n'observe pas une étiquette pour chacune de ses prédictions, mais plutôt une mesure de la qualité de ses prédictions (possiblement prenant en compte toute une séquence de prédictions) [26].

1.4 Généralisation

1.4.1 Sur-apprentissage

L'entraînement d'un algorithme d'apprentissage consiste à extraire, de manière explicite ou implicite, des caractéristiques de la distribution de probabilité P qui génère les données. Mais P étant inconnu, l'algorithme se base à la place sur un nombre fini d'exemples d'entraînement, c'est-à-dire. Sur la distribution discrète $\hat{P}$ des exemples disponibles dans D (appelée la distribution *empirique*). Lorsque certaines caractéristiques apprises sur P ne s'appliquent pas à P, on parle de *sur-apprentissage*, et on risque une mauvaise *généralisation*, c'est-à-dire. Que l'algorithme ne va pas obtenir une bonne performance sur de nouveaux exemples tirés de P. Prenons l'exemple de la classification, lorsqu'un modèle estime $P(y|x)$ par une fonction $q_x(y)$. Afin de mesurer la similarité entre q_x et $P(.|x)$, on peut par exemple utiliser la divergence de Kullback-Leibler D_{KL}[40], définie par [26] :

$$D_{\mathrm{KL}}(P(.|x)||q_x) = \sum_y P(y|x)\, ln\frac{P(y|x)}{q_x(y)} \tag{1}$$

Cette quantité est toujours supérieure ou égale à zéro, et est égale à zéro si et seulement si q_x est égale à$P(.|x)$. La minimisation de la divergence de Kullback-Leibler est donc un critère raisonnable pour obtenir une fonction q_x qui approxime P(·|x). Vu que le but est d'obtenir une bonne approximation pour toutes les valeurs de x qui pourraient être générées par P, il est logique de considérer la minimisation du critère

$$\begin{aligned} C(q) &= E_X[D_{KL}(P(.|x)||q_x)] \\ &= \int_x \sum_y P(x)P(y|x) \ln\frac{P(y|x)}{q_x(y)}\, dx \end{aligned} \tag{2}$$

$C(q)$ Est ici ce que l'on appelle l'erreur de généralisation, c'est-à-dire. L'erreur moyenne sur des exemples tirés de P. Puisque P est inconnue, on minimise en pratique un critère $\hat{C}$ défini de la même manière en remplaçant P par $\hat{P}$. C'est le principe de minimisation du risque empirique [8], et $\hat{C}$ s'écrit ici [26] :

$$\hat{C}(q) = \sum_{i=1}^{n} \frac{1}{n} \ln \frac{1}{q_{x_i}(y_i)} = -\frac{1}{n} \sum_{i=1}^{n} \frac{1}{n} \ln q_{x_i}(y_i) \quad (3)$$

Qui est appelée la *log-vraisemblance négative* (en anglais NLL, pour "Negative Log-Likelihood"). Ce critère est minimisé dès que $q_{x_i}(y_i) = 1$, pour tous les exemples d'entraînement $(x_i, y_i) \in$ D *et ce quelle que soit la valeur de* $q_x(y)$ *pour des valeurs de x non observées* dans D [26].

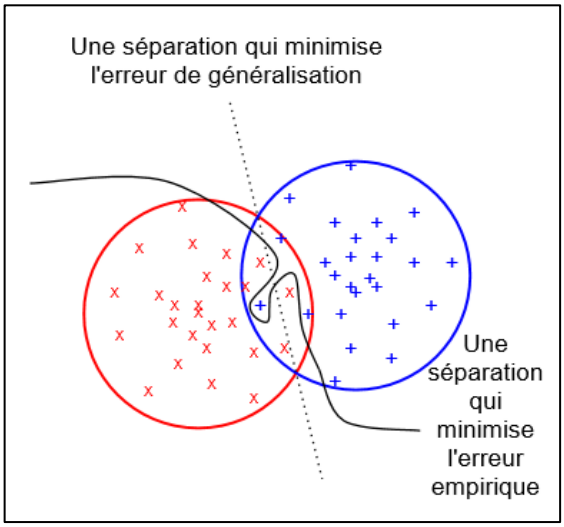


Figure 1. 4 . Situation de sur-apprentissage : classification binaire (les classes sont les cercles rouge et bleu, avec respectivement les × et les + comme exemples d'entraînement). Une séparation idéale en termes d'erreur de généralisation serait la ligne en pointillés, mais un algorithme dont le but est uniquement de minimiser l'erreur empirique pourrait par exemple séparer les exemples selon la ligne pleine : la classification des exemples d'entraînement serait parfaite, mais l'erreur de généralisation serait plus élevée que celle de la ligne en pointillés [26].

Une fonction q peut donc minimiser $\hat{C}(q)$ sans nécessairement minimiser $C(q)$ Si c'est le cas, on est en situation de sur-apprentissage, illustrée en figure 1.4. Pour une distribution fixe des données, deux facteurs principaux augmentent le risque de sur-apprentissage [26] :

– Le manque d'exemples d'entraînement : moins il y a d'exemples, plus il existe de fonctions minimisant le critère $\hat{C}(q)$(éq. 3), parmi lesquelles seulement un petit nombre seront vraiment proches de la "vraie" solution au problème.

– Pas assez de contraintes dans la forme de la fonction q : moins la classe de fonctions à laquelle q appartient est restreinte, plus l'algorithme d'apprentissage risque de tirer parti de la fléxibilité de q pour apprendre des "détails" des exemples d'entraînement, qui

ne se généralisent pas à la vraie distribution *P*. C'est le cas par exemple dans la figure 1.4 où la séparation tarabiscotée des exemples par la ligne pleine permet de minimiser l'erreur empirique, mais va mener à une plus grande erreur de généralisation qu'une simple ligne droite [26].

1.4.2 Régularisation

Un moyen de lutter contre le sur-apprentissage est d'utiliser une technique dite de régularisation. Il existe plusieurs méthodes de régularisation, mais elles partagent le même principe : rajouter au processus d'apprentissage des contraintes qui, si elles sont appropriées, vont améliorer les capacités de généralisation de la solution obtenue.

Reprenons par exemple le cas de la classification, où l'on cherche à minimiser le critère $C(q)$(éq. 2), que l'on approxime par $\hat{C}(q)$(éq. 3). Comme nous venons de le voir, ce problème est mal défini car il existe une infinité de fonctions qui minimisent C, sans donner aucune garantie sur la valeur de C. Une première façon de régulariser le problème est donc de restreindre la forme de q : par exemple $x \in \mathbb{R}^d$et $y \in \{0,1\}$ on peut se limiter aux fonctions de la forme [26].

$$q_x(1) = \frac{1}{1+e^{-w^T x}} \tag{4}$$

Où $w \in \mathbb{R}^d$ est le vecteur de paramètres du modèle. Notons que si les x_i de l'ensemble d'entraînement sont linéairement indépendants, alors cette contrainte sur la forme de q n'est pas suffisante, puisqu'il est toujours possible que $\hat{C}(q)$soit arbitrairement proche de zéro sans pour autant avoir de garantie sur la valeur de $C(q)$. Une technique classique de régularisation consiste alors à rajouter au critère $\widehat{C}$ une mesure qui pénalise la *complexité* de la solution, suivant le principe du rasoir d'Occam qui dit que les hypothèses les plus simples sont les plus vraisemblables voir [3]. Une possibilité est de minimiser [26].

$$\tilde{C}(q) = \hat{C}(q) + \lambda||w||^2 \tag{5}$$

Au lieu de $\hat{C}$, pour q défini comme dans (l'éq. 4), afin d'empêcher le vecteur w de contenir des valeurs arbitrairement grandes (en valeur absolue). Le paramètre λ contrôle la force de cette contrainte (lorsque $\lambda \to +\infty$ la seule solution possible est la fonction constante $q_x(1) = q_x(0) = 0{,}5$, qui est la plus simple qu'on puisse imaginer). Le critère empirique $\hat{C}(q^*)$pour la fonction q^* qui minimise le critère régularisé $\tilde{C}$ pourrait ne pas être proche de zéro, mais on peut souvent ainsi – pour certaines valeurs de λ – obtenir des valeurs plus basses du critère de généralisation

C (celui qui nous intéresse vraiment). C'est le principe de la minimisation du risque structurel [8] [26].

Dans cet exemple, nous avons utilisé $||w||^2$ pour mesurer la complexité de la fonction q définie à partir de w par (l'éq. 4). En général, il n'existe pas une seule mesure de complexité universelle qui soit appropriée pour tous les algorithmes d'apprentissage, et le choix de la mesure de complexité à pénaliser joue un rôle très important. La complexité de Kolmogorov [7][4], est une mesure de complexité très générique qui est intéressante en théorie, même si en pratique elle est souvent impossible à utiliser directement. Elle consiste à dire que la complexité d'une fonction est la taille du plus petit programme qui l'implémente. Un premier obstacle à l'utilisation de cette complexité est le fait qu'il faille choisir un langage de programmation approprié : par exemple si le langage choisi contient une fonction primitive qui calcule le produit scalaire, alors, dans notre exemple ci-dessus la plupart des fonctions q définies par (l'éq. 4) ont la même complexité de Kolmogorov. Par contre, si le produit scalaire n'est pas une primitive du langage (et qu'il n'y a pas d'instruction de boucle), alors il faut l'écrire comme une somme de produits et q est d'autant plus complexe que w a d'éléments non nuls. Une autre difficulté est qu'il n'est en général pas possible d'optimiser la complexité de Kolmogorov de manière efficace, ce qui rend vaine son utilisation directe dans un processus d'optimisation. Elle a malgré tout de nombreuses applications, comme décrit dans le livre de [5] [26].

1.4.3 Malédiction de la dimensionnalité

On peut observer empiriquement – et dans certains cas justifier mathématiquement –que plus la dimension d de l'entrée x est élevée, plus les tâches d'apprentissage machine ont tendance à être difficiles à résoudre. C'est ce qu'on appelle *la malédiction de la dimensionalité* [1]. Il existe plusieurs manifestations de cette malédiction. La plus importante dans le contexte de cette thèse est le fait que le nombre de combinaisons possibles des entrées augmente exponentiellement avec la dimensionnalité d : en notant x_{ij} la valeur associée à la j-ème dimension de l'entrée x_i, si l'on suppose que ces entrées ne peuvent prendre qu'un nombre fini k de valeurs, alors le nombre de combinaisons possibles est égal à k^d. Un algorithme qui apprend "bêtement" à associer une valeur à chaque combinaison sans partager d'information entre les différentes combinaisons n'a aucune chance de fonctionner en haute dimension, car il ne pourra pas généraliser aux multiples combinaisons qui n'ont pas été vues dans l'ensemble d'entraînement. Dans le cas où x_{ij} n'est pas contraint dans un ensemble fini de valeurs, l'intuition reste la même pour certains algorithmes qui consistent à "partitionner" $\mathbb{R}^d$ en régions

indépendantes (possiblement de manière implicite) : si le nombre de ces régions augmente exponentiellement avec *d*, alors un tel algorithme aura de la difficulté à généraliser pour de grandes valeurs de *d*. La figure 1.5 illustre ce phénomène en une et deux dimensions, et il faut garder à l'esprit que la situation peut s'avérer encore bien pire lorsque l'on manipule des entrées à plusieurs centaines de dimensions [26].

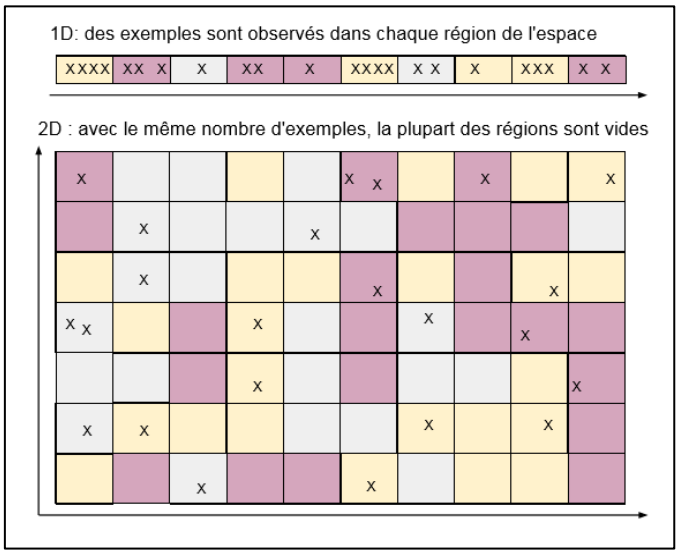


Figure 1. 5 . Malédiction de la dimensionalité : si l'algorithme partitionne l'espace en régions indépendantes, le nombre d'exemples nécessaires pour remplir ces régions augmente de manière exponentielle avec la dimension. Ici, la couleur d'une région représente la classe majoritaire dans cette région, et un tel algorithme pourrait bien généraliser à partir de 23 exemples d'entraînement pour le problème du haut (1D), mais pas pour celui du bas (2D) [26].

1.5 Différents types de modèles

1.5.1 Modèles paramétriques

En apprentissage machine, un modèle paramétrique est défini par un ensemble Θ de paramètres de dimension fini, et l'algorithme d'apprentissage associé consiste à trouver la meilleure valeur possible de Θ. La figure 1.6 montre un exemple de régression linéaire en une dimension, sans régularisation.

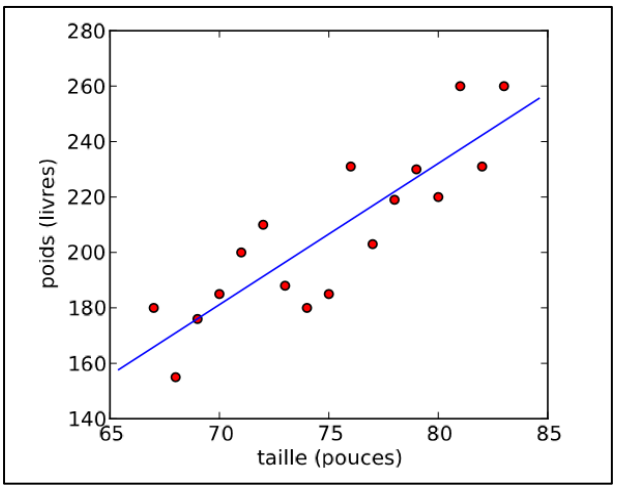


Figure 1. 6 . Modèle paramétrique : la régression linéaire. Les données (points rouges) sont un sous-ensemble des mêmes données que dans la figure 1.2, et la tâche est ici de prédire le poids d'un joueur de baseball en fonction de sa taille [26].

Les modèles paramétriques peuvent également être statistiquement inefficaces. S'il y a moins d'exemples d'apprentissage, alors le problème est sur-paramétré et on risque le sur-apprentissage : le modèle pourrait apprendre des paramètres taillés "sur mesure" pour les données d'entraînement, mais qui mèneront à une mauvaise généralisation. La conséquence de cette observation est qu'en général, un modèle avec un grand nombre de paramètres est statistiquement inefficace [26].

1.5.2 Modelés non paramétriques

Un modèle non paramétrique n'a au contraire pas d'ensemble exacte de paramètres : le nombre de variables utilisées par le modèle augmente généralement avec le nombre d'exemples dans l'ensemble d'entrainement. Un exemple de modèle non paramétrique pour résoudre le même problème de régression que celui décrit en 4.1 est l'algorithme des fenêtres de Parzen,

aussi appelé régression de Nadaraya-Watson [6][9]. La figure 1.7 montre un exemple de régression par fenêtres de Parzen en une dimension [26].

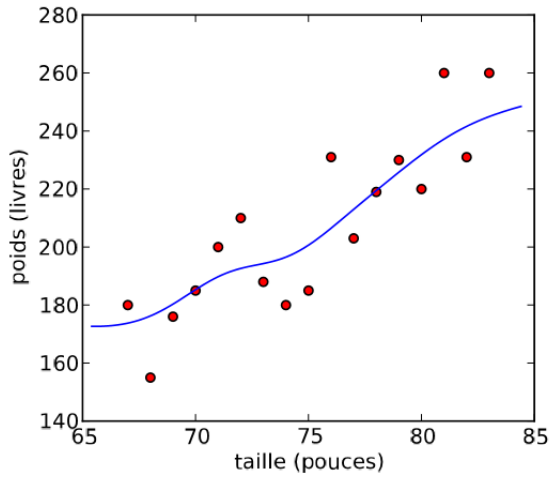


Figure 1. 7 . Modèle non paramétrique : régression par fenêtres de Parzen. Les données sont les mêmes que dans l'exemple de la régression linéaire (La figure 1.6) [26].

1.6 Algorithmes d'apprentissage

1.6.1 Réseaux de neurones

Comme leur nom l'indique, les réseaux de neurones sont inspirés de l'architecture du cerveau, étant organisés en couches de neurones connectées entre elles. La première couche, appelée la couche d'entrée, est de la même dimension que les entrées x et l'activation de son j-ème neurone (c'est-à-dire la valeur qu'il calcule) est égale à x_{ij} lorsque l'on calcule la prédiction du réseau sur l'exemple x_i. La dernière couche, appelée la couche de sortie, est de la même dimension que l'étiquette dans le cas d'une tâche supervisée. Par exemple, pour la classification, le j-ème neurone de la couche de sortie va calculer $P(Y = j|x_i)$ lorsque x_i est dans la couche d'entrée. Les couches intermédiaires sont appelées les couches cachées. Il existe de nombreuses variations dans les architectures de réseaux de neurones. L'architecture la plus connue est celle où il existe une unique couche intermédiaire h qui calcule une transformation non linéaire des entrées, de la forme [26] :

$$H(x) = \sigma(W^T x + b) \quad (7)$$

Où W est une matrice de poids, b est un vecteur de biais (de la même taille que le nombre de neurones dans la couche cachée), et σ est une transformation non linéaire élément par élément, dont l'opération sur chaque élément est typiquement la sigmoïde ou la tangente hyperbolique [26]

$$\text{Sigmoid(u)} = \frac{1}{1+e^{-u}} \tag{8}$$

$$\tanh(u) = \frac{e^{u}-e^{-u}}{e^{u}+e^{-u}} \tag{9}$$

La fonction donnant la sortie $\hat{y}$ en fonction de h dépend de la tâche ; en classification, on écrit souvent le $j^{éme}$ neurone de $\hat{y}$, qui estime $P(Y = j|x)$,sous la forme :

$$\frac{e^{V_j^T h+c_j}}{\sum_k e^{V_k^T h+c_k}} \tag{10}$$

Avec V_j la j-ème rangée d'une matrice de poids V, et c le biais du j-ème neurone de sortie. La figure 1.8 montre une telle architecture à une couche cachée.

L'apprentissage dans un réseau de neurones consiste à optimiser les poids et biais de façon à minimiser un coût approprié à la tâche. Par exemple, pour la classification, on minimisera la log-vraisemblance négative empirique (éq. 3) qui, en notant Θ les paramètres à optimiser, et f_j(xᵢ; Θ) la valeur du j-éme neurone de sortie lorsque x_i est en entrée du réseau, se réécrit [26] :

$$\hat{C}(\Theta) = -\frac{1}{n}\sum_{i=1}^{n} \ln f_{y_i}(x_i;\Theta) \tag{11}$$

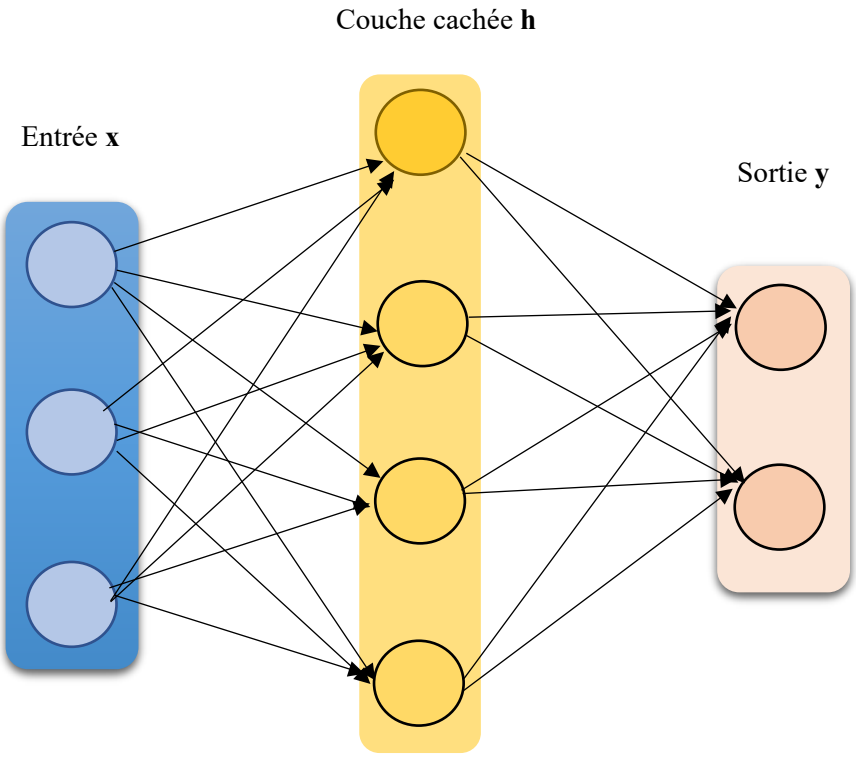


Figure 1. 8 . Réseau de neurones à une couche cachée. Une flèche du neurone *i* vers le neurone *j* indique que l'activation de j dépend directement de celle de *i* [26].

Un autre exemple, en apprentissage non supervisé, est celui des réseaux de neurones auto-associateurs dont le but est d'extraire dans la couche cachée une représentation qui permet de reconstruire l'entrée x_i (donc l'étiquette y_i est en fait égale à x_i). Dans ce cas, le coût le plus fréquemment utilisé est l'erreur de reconstruction quadratique moyenne [26]

$$\hat{C}(\Theta) = \frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{d}(f_i(x_i;\Theta) - x_{ij})^2 \qquad (12)$$

Mais lorsque les entrées $x_{ij} \in \{0,1\}$, on peut également minimiser l'entropie croisée

$$\hat{C}(\Theta) = -\frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{d}((x_{ij} ln f_i(x_i;\Theta) + \left(1 - x_{ij}\right)\ln(1 - f_i(x_i;\Theta) \qquad (13)$$

Où le terme dans la somme peut s'interpréter comme la log-vraisemblance des données d'entraînement selon la distribution $P_\Theta(X_{ij} = 1|x_i) = f_i(x_{ij};\Theta)$.

L'apprentissage dans un réseau de neurones consiste à optimiser les poids et biais de façon à minimiser un coût approprié à la tâche. Les algorithmes d'optimisation les plus souvent

utilisés pour minimiser $\hat{C}(\Theta)$ sont des algorithmes d'optimisation locale basés sur l'idée de *la descente de gradient* : le gradient du coût $\hat{C}(\Theta)$ par rapport aux paramètres Θ, noté $\frac{\partial \hat{C}(\Theta)}{\partial \Theta}$ indique la direction dans laquelle le coût augmente le plus lorsque Θ varie. Descendre le gradient signifie déplacer les paramètres Θ dans la direction opposée, de manière à diminuer le coût. Le gradient par rapport à tous les paramètres du réseau peut se calculer de manière efficace grâce à l'algorithme de rétro-propagation du gradient [21] [26].

En apprentissage non supervisé, on parle des réseaux de neurones auto-associateurs dont le but est d'extraire dans la couche cachée une représentation qui permet de reconstruire l'entrée x_i (donc l'étiquette y_i est en fait égale à x_i). Dans ce cas, le coût le plus fréquemment utilisé est l'erreur de reconstruction quadratique moyenne. On trouve donc que les réseaux de neurones représentent l'algorithme le plus approprié pour notre projet et pour l'implémentation [26].

1.6.2 Machines de Boltzmann restreintes

Dans sa version la plus simple, une machine de Boltzmann restreinte (RBM)est un algorithme non supervisé qui modélise la distribution $P(X)$ où $x \in \{0,1\}^d$ comme la marginale d'une distribution jointe [26]

$$P(x,h) = \frac{e^{-\varepsilon(x,h)}}{\mathrm{Z}} \tag{14}$$

Où Z est *la fonction de partition* assurant que cette probabilité est bien normalisée :

$$Z = \sum_{x,h} e^{-\varepsilon(x,h)} \tag{15}$$

Le vecteur $h \in \{0,1\}^k$ représente la couche cachée, tandis que le vecteur *x* est appelée la couche visible. Les éléments d'une couche sont généralement appelés unités plutôt que neurones. La quantité $\varepsilon(x,h)$ est *l'énergie* de la paire (x,h), et l'équation 13 montre que plus l'énergie est faible, plus cette paire est probable. Une forme typique pour ε est [26] :

$$\varepsilon(x,h) = h^T W x - x^T b - h^T c \tag{16}$$

Où la matrice $W \in \mathbb{R}^{k \times d}$ et les vecteurs b $\in \mathbb{R}^d$ et c $\in \mathbb{R}^k$ sont les paramètres du modèle. Bien que ce modèle ait originellement été introduit sous un autre nom [44], il s'agit bien d'une forme particulière de machine de Boltzmann [16][15]. Comparée à une machine de Boltzmann générique, l'énergie d'une RBM à la propriété que les probabilités conditionnelles $P(x|h)$ et $P(h|x)$ se factorisent, ce qui rend l'entraînement (un peu) plus aisé. Cette propriété se visualise lorsque l'on représente une RBM sous la forme d'un modèle graphique non dirigé [25], comme

dans la figure 1.9 : il y a des connexions entre les unités visibles et les unités cachées, mais aucune connexion de visible à visible, ni de cachée à cacher [26].

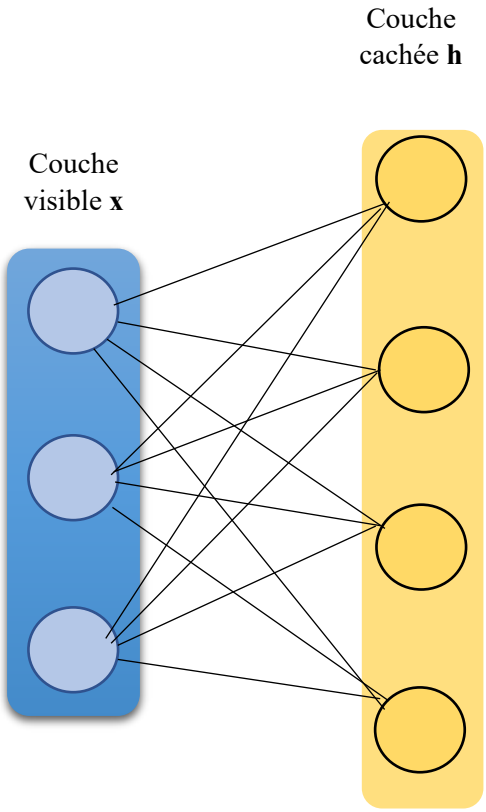


Figure 1. 9 . Machine de Boltzmann restreinte [26].

1.6.3 Architectures profondes (Deep Learning)

Le terme d'architecture profonde désigne toute une famille de modèles inspirés des réseaux de neurones, dont le point commun est la composition de transformations successives, permettant de calculer une fonction complexe de l'entrée. Par exemple, un réseau de neurones avec une seule couche intermédiaire peut être considéré comme une architecture profonde si l'on rajoute des couches cachées : chaque couche additionnelle augmente la profondeur du réseau, et déterminer la profondeur idéale en fonction des données fait partie de la problématique des algorithmes d'apprentissage pour architectures profondes. Les réseaux de ce type ont longtemps été ignorés, d'une part parce qu'ils se sont avérés beaucoup plus difficiles à optimiser que les réseaux à une seule couche cachée, d'autre part parce qu'il a été démontré que les réseaux à une seule couche sont des approximateurs universels [17] [26]. L'intérêt pour les réseaux profonds est récemment réapparu après avoir découvert qu'une initialisation non

supervisée des poids du réseau peut mener à de bien meilleures performances que l'initialisation aléatoire utilisée jusqu'à présent.

1.6.4 K-plus proches voisins

L'algorithme des k plus proches voisins est un algorithme non paramétrique utilisé pour la régression et la classification. Etant donnée une mesure de distance dans l'espace d'entrée $\mathbb{R}^d$ (souvent prise comme la distance Euclidienne$||x_i - x_j||$) la prédiction du modèle sur un exemple de test $x \in T$ dépend uniquement des k plus proches voisins de x dans l'ensemble d'entraînement D. En notant $i_1(x), \ldots, i_k(x)$ les indices des k exemples de D les plus proches de x selon la distance choisie (ses "voisins"), la prédiction du modèle en régression est la moyenne des étiquettes observées chez ces k voisins [26] :

$$f(x) = \frac{1}{k}\sum_{j=1}^{k} y_{i_j(x)} \tag{17}$$

Et en classification il s'agit d'un vote parmi les k voisins :

$$f(x) = argmax_y \sum_{j=1}^{k} 1_{y=y_{i_j(x)}} \tag{18}$$

Où en cas d'égalité parmi les votes le modèle choisit aléatoirement l'une des classes majoritaires. La classification par les k plus proches voisins est illustrée en figure 1.11.

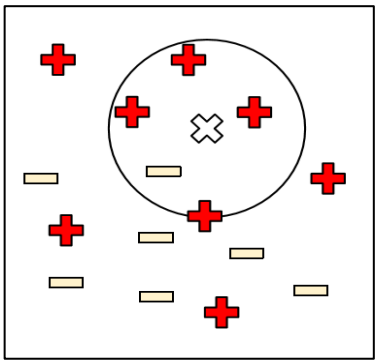


Figure 1. 10 . K-plus proches voisins (k = 5, tâche de classification) [26].

1.6.5 Fenêtres de Parzen

L'algorithme des fenêtres de Parzen a déjà été présenté dans le contexte de la régression non paramétrique, où on l'appelle parfois la régression à noyau ou la régression de Nadaraya et Watson [6][9]. On peut également utiliser une approche similaire en apprentissage non supervisé pour l'estimation de densité [20] [18], en estimant la densité de probabilité au point *x* par [26] :

$$f(x) = \frac{1}{n}\sum_{i=1}^{n} K(x, x_i) \tag{19}$$

Ce qui correspond à placer une masse de probabilité "autour" de chaque exemple d'apprentissage x_i, dans un volume défini par le noyau K. Ici, K doit respecter les contraintes [26]

$$K(x, x_i) \geq 0$$

$$\int_x K\,(x, x_i)dx = 1 \tag{20}$$

De manière à ce que *f* soit une densité de probabilité valide. Le choix le plus répandu pour K est le noyau Gaussien, mais avec la normalisation appropriée [26] :

$$K(x, x_i) = \mathcal{N}\left(x_i; x_j, \sigma^2\mathbf{I}\right) = \frac{1}{(2\pi)^{d/2}\sigma d}\, e^{-\frac{||x_i - x_j||^2}{2\sigma^2}} \tag{21}$$

où l'on note $\mathcal{N}(\cdot; \mu, \Sigma)$ la densité de probabilité d'une Gaussienne de moyenne μ et covariance Σ. Un exemple d'estimation de densité par fenêtres de Parzen avec un noyau Gaussien est montré en figure 1.12 [26].

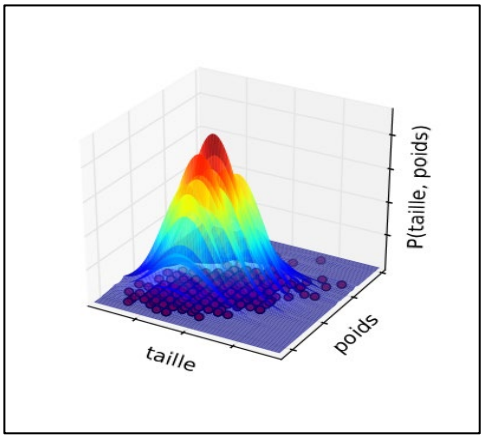


Figure 1. 11 . Fenêtres de Parzen pour l'estimation de densité [26].

1.6.6 Mélanges de Gaussiennes

Les mélanges de Gaussiennes généralisent les fenêtres de Parzen (avec noyau Gaussien) pour l'estimation de densité, en écrivant la densité comme une somme pondérée de Gaussiennes [52] :

$$f(x) = \sum_{j=1}^{N} \alpha_j \mathcal{N}(x; \mu_j, \Sigma_j) \quad (22)$$

Où N est le nombre de composantes du mélange, et α_j le poids de la j^{ème} composante (les poids sont tels que $\alpha_j \geq 0$ et$\Sigma_j \alpha_j = 1$). L'interprétation dite "générative'' de cette équation est que le modèle suppose que chaque exemple observé a été généré de la façon suivante [26] :

1.Une composante j est choisie aléatoirement, avec probabilité α_j.

2.Un exemple est généré par une distribution Gaussienne centrée en μ_j avec covariance Σj.

Si N = n, $\alpha_j = n^{-1}$, $\mu_j = x_i$ et $\Sigma_j = \sigma^2\mathbf{I}$ on retrouve les fenêtres de Parzen non paramétriques vues précédemment. Mais on peut également apprendre un modèle paramétrique de mélange en fixant N et en optimisant les poids α_j, les centres μ_j et les covariances Σ_j de chaque Gaussienne. L'algorithme le plus populaire pour l'apprentissage d'un mélange de Gaussiennes est l'algorithme *Espérance-Maximisation* (EM) [14].

1.6.7 Méthodes à noyau

Plusieurs algorithmes mentionnés précédemment utilisent une fonction noyau $K(x_i, x_j)$ pour mesurer la similarité entre deux entrées x_i et x_j. Les méthodes dites "à noyau" incluent en particulier une catégorie d'algorithmes basés sur ce que l'on appelle "l'astuce du noyau", qui s'applique à tout algorithme qui peut s'exprimer uniquement à partir de produits scalaires de la forme $x_i^T x_j$. Cette astuce consiste à remplacer $x_i^T x_j$ dans l'algorithme d'origine par une fonction noyau $K(x_i, x_j)$, où K doit satisfaire certaines propriétés mathématiques (on dit que le noyau est défini positif – c'est le cas par exemple du noyau Gaussien que nous avons déjà utilisé). Cela revient à appliquer l'algorithme sur les données transformées implicitement et de manière non linéaire par une fonction ϕ telle que $\varphi(x_i)^T \varphi(x_j) = K(x_i, x_j)$ (une telle fonction ϕ existe automatiquement si K est défini positif, et en pratique on n'a pas besoin de la calculer explicitement). L'intérêt du noyau est principalement d'effectuer une extraction de caractéristiques non linéaire à partir des entrées, ce qui peut améliorer les performances de l'algorithme [26].

La plus célèbre méthode à noyau exploitant cette astuce est l'algorithme de la machine à vecteurs de support (SVM, pour "Support Vector Machine" en anglais). Il s'agit d'un algorithme de classification basé sur l'idée de marge : pour obtenir une meilleure généralisation, il est préférable de laisser une marge entre les exemples et la surface de décision [11][13][24]. Ce concept de marge est illustré par la figure 1.17 : dans le cas linéaire (c'est à dire. Sans utiliser l'astuce du noyau) la marge du classifieur est la distance entre l'hyper-plan séparant les deux classes et les exemples les plus proches. Dans le cas non linéaire, le même principe s'applique dans l'espace des exemples projetés implicitement par ϕ, ce qui se traduit dans l'espace des entrées par une séparation non linéaire des exemples de chaque classe. Depuis les SVMs, l'astuce du noyau et l'idée de marge ont inspiré un grand nombre de nouveaux algorithmes, ainsi que la "noyautisation" d'algorithmes existants [22] [26].

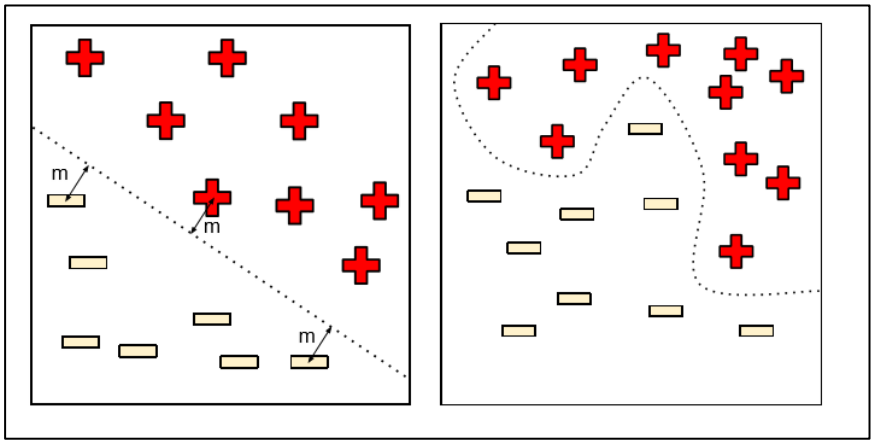


Figure1.13. Machine à vecteurs de support, dans le cas linéaire (à gauche) et non linéaire (à droite) [26].

1.6.8 Arbres de décision

Les arbres de décision forment une famille d'algorithmes d'apprentissage utilisés pour la classification et la régression [12]. Un arbre de décision est un arbre où chaque nœud interne représente un test sur l'entrée x_i. L'exemple typique d'un arbre de décision est un arbre binaire où le test effectué à chaque nœud k est de la forme $x_{ij}<\theta_k$, c'est-à-dire. Qu'on compare la $j^{éme}$ coordonnée de x_i à un seuil θ_k : si elle est plus petite, on continue de parcourir l'arbre en suivant la première branche du nœud, sinon on suit la seconde branche. Lorsqu'on atteint finalement une feuille de l'arbre (un nœud sans enfants), on dit que l'exemple x_i appartient à cette feuille. La figure 1.18 montre un tel arbre de décision [26].

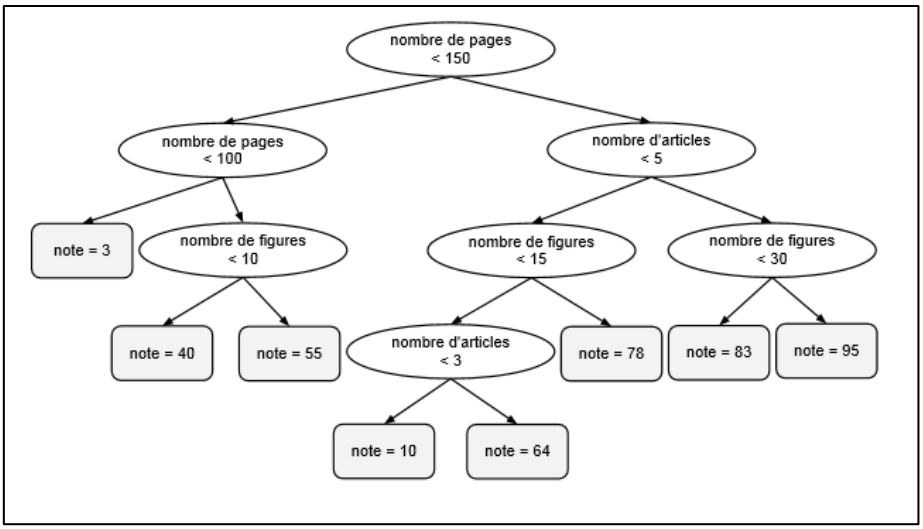


Figure1.18. Arbre de décision typique [26]

L'apprentissage de ce type d'arbre de décision consiste à choisir les variables testées à chaque nœud, les seuils de comparaison, la profondeur de l'arbre, ainsi que la fonction de décision associée à chaque feuille. Il existe plusieurs algorithmes pour cela, mais leur principe de base est de faire en sorte que le résultat du test effectué à chaque nœud donne de l'information supplémentaire sur $P(Y|x)$. Idéalement, la distribution $P(Y|x)$pour un nouvel exemple *x* doit être bien approximée par la distribution empirique des étiquettes des exemples d'entraînement appartenant à la même feuille que *x* [26].

1.6.9 Méthodes Bayésiennes

Les méthodes Bayésiennes tirent leur nom de la célèbre règle de Bayes [26] :

$$P(\Theta|\mathcal{D}) = \frac{P(\Theta|\mathcal{D})P(\Theta)}{P(\mathcal{D})} \tag{28}$$

Qui est ici appliquée avec d'une part les paramètres Θ d'un modèle, et d'autre part les données d'entraînement D observées. La quantité $P(\Theta|D)$ est appelée la vraisemblance : c'est la probabilité d'observer les données D en supposant qu'elles ont été générées par notre modèle dont les paramètres sont Θ. P(Θ) est appelée la distribution a priori : c'est une distribution sur l'espace des paramètres qui reflète notre croyance sur les valeurs possibles des paramètres avant l'observation des données. P(Θ|D) est alors calculée comme étant proportionnelle au produit de la vraisemblance par la distribution a priori : elle est appelée la distribution a posteriori, c'est-à-dire. Qu'elle indique la probabilité des paramètres après avoir observé les données. Le terme P(D) est un terme de normalisation qui peut se calculer, si nécessaire, par $P(D) = \int_{\Theta} P(D|\Theta)P(\Theta)d\Theta$. Il suffira de garder à l'esprit qu'il s'agit de méthodes probabilistes, qui ont

l'avantage en particulier de prendre en compte l'incertitude de manière naturelle : par exemple, la prédiction d'un modèle Bayésien supervisé sur une nouvelle entrée *x* peut s'écrire comme [26]

$$P(y|x, D) = \int_{\Theta} P(\Theta)P(\Theta|D)d\Theta \quad (29)$$

Et la variance de cette distribution peut être interprétée comme l'incertitude sur la prédiction de l'étiquette *y*. Notons qu'il n'est en général pas trivial de calculer une telle intégrale, et que plusieurs techniques ont été développées spécifiquement dans ce but [10] [26].

1.7 Conclusion

Dans ce chapitre nous avons défini la notion de l'apprentissage artificiel et ses caractéristiques et nous avons introduit ses composants. Nous avons appris ses avantages. Ainsi, nous avons présenté quelques travaux antérieurs et récents ayant un lien avec notre travail de recherche.

Nous présenterons dans le chapitre suivant le domaine de reconnaissance des formes et ses caractéristiques principales

Chapitre 02 : Reconnaissance des formes.

2.1 Introduction

Dans ce deuxième chapitre, nous définissons le domaine de reconnaissance de formes et ses caractéristiques principales et nous introduisons les différents types des formes (motifs, pattern) traité par cette branche.

2.2 Reconnaissance des formes (RF)

2.2.1 Définition

On désigne par **reconnaissance de formes** (ou parfois **reconnaissance de motifs**) un ensemble de techniques et méthodes visant à identifier des motifs à partir de données brutes afin de prendre une décision dépendant de la catégorie attribuée à ce motif. On considère que c'est une branche de l'intelligence artificielle qui fait largement appel aux techniques d'apprentissage automatique et aux statistiques [39].

Les formes à reconnaître peuvent être de nature très variée. Il peut s'agir de contenu visuel (code barre, visage, empreinte digitale, texte.....) Ou sonore (reconnaissance de parole), d'images médicales (rayon X, EEG, IRM…) ou multi spectrales (images satellitaires) et bien d'autres [39].

Watanabe [27] a défini une forme comme : *« l'opposé du chaos ; c'est une entité vaguement définie, à laquelle on peut associer un nom* ». En des termes informatiques, une forme est un ensemble de valeurs, appelés attributs, auxquels est associé un nom (ou étiquette), qui est leur classe. Plusieurs formes peuvent avoir la même classe, on dit alors que ce sont les exemples ou réalisations de la classe [39].

Le problème que cherche à résoudre la reconnaissance des formes est d'associer une classe à une forme inconnue (qui n'a pas encore de classe associée). On considère souvent la Reconnaissance des formes comme un problème de classification : trouver la fonction qui affecte à toute forme inconnue sa classe la plus pertinente. Elle est partie intégrante de tout système intelligent destine à la prise de décision [28] [39].

2.2.2 Historique

Or que ce soit pour déchiffrer un texte dactylographié ou manuscrit, pour compter des chromosomes, reconnaître une tumeur, un char ou un avion de guerre, la compréhension de l'image, sa classification passe toujours par la reconnaissance d'une forme. « *Plusieurs approches théoriques ont été développées* », explique *Olivier Faugeras* [39].

« Les premières consistaient à faire des calculs à partir de l'image et construire des représentations symboliques de plus en plus complexes, d'abord en deux dimensions tel que sur l'image, puis tridimensionnelles, pour tenter de restituer une description proche de notre propre vision. » Un peu partout dans le monde, les chercheurs ont mis au point des méthodes mathématiques permettant de détecter les contours des objets à partir des changements rapides de contraste dans l'image, des ombres et des lumières, des régions homogènes en couleur, en intensité, en texture [39].

« Dès 1964, des chercheurs français, Georges Matheron (1930-2000) et Jean Serra, ont développé une autre approche théorique (baptisée morphologie mathématique) et un outil spécifique (l'analyseur de texture breveté en 1965, ndlr) d'abord pour analyser des microphotographies de terrain et évaluer des teneurs en minerai, puis pour d'autres applications comme la cytologie (caractérisation et comptage de cellules) » rappelle Olivier Faugeras. En 1968, ils créent le Centre de morphologie mathématique de l'Ecole des Mines de Fontainebleau. Leurs outils d'analyse et d'interprétation d'images sont longtemps restés franco-français, jusqu'à ce qu'un américain, Robert Haralick (Université du Kansas à cette époque, de Seattle actuellement), en fasse une large publicité dans les années 1980, en les adaptant à de nombreuses applications : industrielles comme l'inspection radiographique des ailes d'avions de Boeing, aériennes ou médicales [38] [39].

D'autres chercheurs, comme les américains Marvin Minsky et Seymour Papert du MIT (Massachussets Institute of Technology) ont considéré le problème dans l'autre sens, en cherchant à formaliser et à faire reproduire par l'ordinateur notre propre processus de reconnaissance d'images, donc notre propre vision. Cette démarche était dans l'air du temps, au cœur des promesses de « l'intelligence artificielle » qui devait permettre de mettre l'intelligence en équations et doter les ordinateurs de toutes les capacités humaines de raisonnement, mémoire, perception. Or la vision s'est révélée être un domaine particulièrement complexe à modéliser tant elle est basée sur une quantité phénoménale de connaissances à priori fondée sur notre intelligence et notre expérience [29] [39].

2.3 Méthodes

La reconnaissance de motifs peut être effectuée au moyen de divers algorithmes d'apprentissage automatique tels : un réseau de neurones. Les formes recherchées peuvent être des formes géométriques, descriptibles par une formule mathématique, telles que : cercle ou ellipse, courbes de Bézier, splines, droite. Elles peuvent aussi être de nature plus complexe telles que : Lettre, chiffre, empreinte digitale.

Les algorithmes de reconnaissance peuvent travailler sur des images en noir et blanc, avec en blanc les contours des objets se trouvant dans l'image. Ces images sont le fruit d'algorithmes de détection de contours. Ils peuvent aussi travailler sur des zones de l'image prédéfinies issues de la segmentation de l'image [39].

2.4 La reconnaissance de plusieurs objets dans une image

Une seule image peut être constituée d'un ou plusieurs objets [30]. Dans le cas où on désire détecter plusieurs objets dans une même image, on peut utiliser le procédé de reconnaissance multi objets représenté sur la figure 2.1 ci- dessous [39].

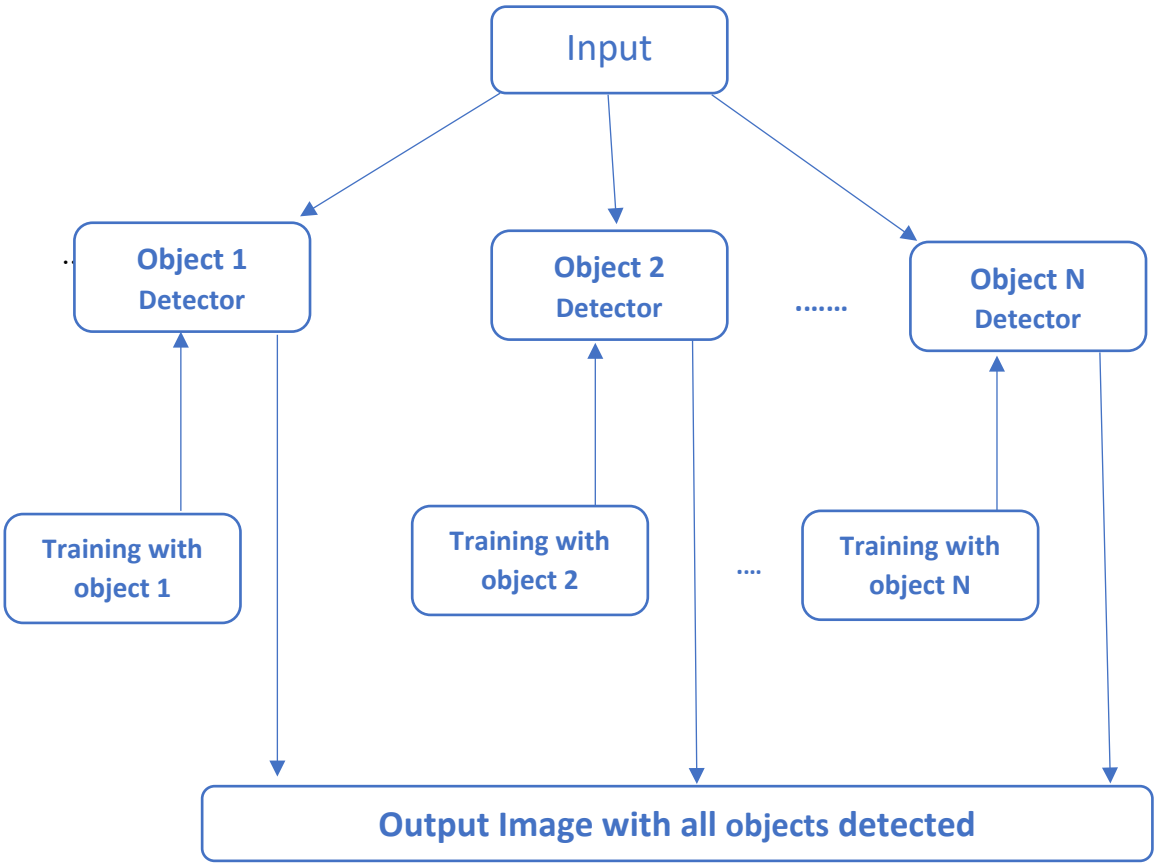


Figure 2. 1. La méthode de reconnaissance de multi objets dans une image.

2.5 Application Typique de la reconnaissance des formes

2.5.1 Marketing

La reconnaissance des formes est souvent utilisée pour classer les consommateurs selon les produits qu'ils sont susceptibles d'acheter. Elle est aussi utilisée par les sociétés de vente pour classer les clients selon qu'ils soient de bons ou mauvais payeurs, ou encore selon qu'ils vont oui ou non passer à la concurrence [31] [39].

2.5.2 Finances

Les systèmes de reconnaissance des formes sont utilisés pour la détection de transactions bancaires frauduleuses ainsi que la prédiction des banqueroutes [32] [39].

2.5.3 Usinage

La qualité des produits dépend souvent de paramétrisation correcte, et les relations exactes entre la qualité et les valeurs des paramètres n'est pas claire. Les systèmes de reconnaissance des formes sont utilisés pour classer les paramètres selon la qualité des produits qu'ils sont susceptibles de générer. Ils permettent ainsi de réduire le nombre d'essais ce qui fait gagner du temps et de l'argent [33] [39].

2.5.4 Energie

Les systèmes de reconnaissance des formes sont utilisés pour prévoir la consommation électrique (réduite, normale, élevée), permettant ainsi aux clients de réduire si nécessaire leur consommation, et aux producteurs de mieux gérer leurs unités de production [34] [39].

2.5.5 Lecture automatisée

Les systèmes de reconnaissance des formes permettent de numériser les anciens documents ainsi que les archives, non pas sous la forme d'images, mais plutôt sous une forme textuelle [35] [39].

2.5.6 Sécurité

La reconnaissance vocale et rétinienne est un exemple d'applications typiques de la reconnaissance des formes pour l'authentification. La vérification des signatures est aussi très populaire [36] [39].

2.6 Schéma général d'un système de Reconnaissance des Formes

La majorité des systèmes de RF ont le schéma de fonctionnement suivant :

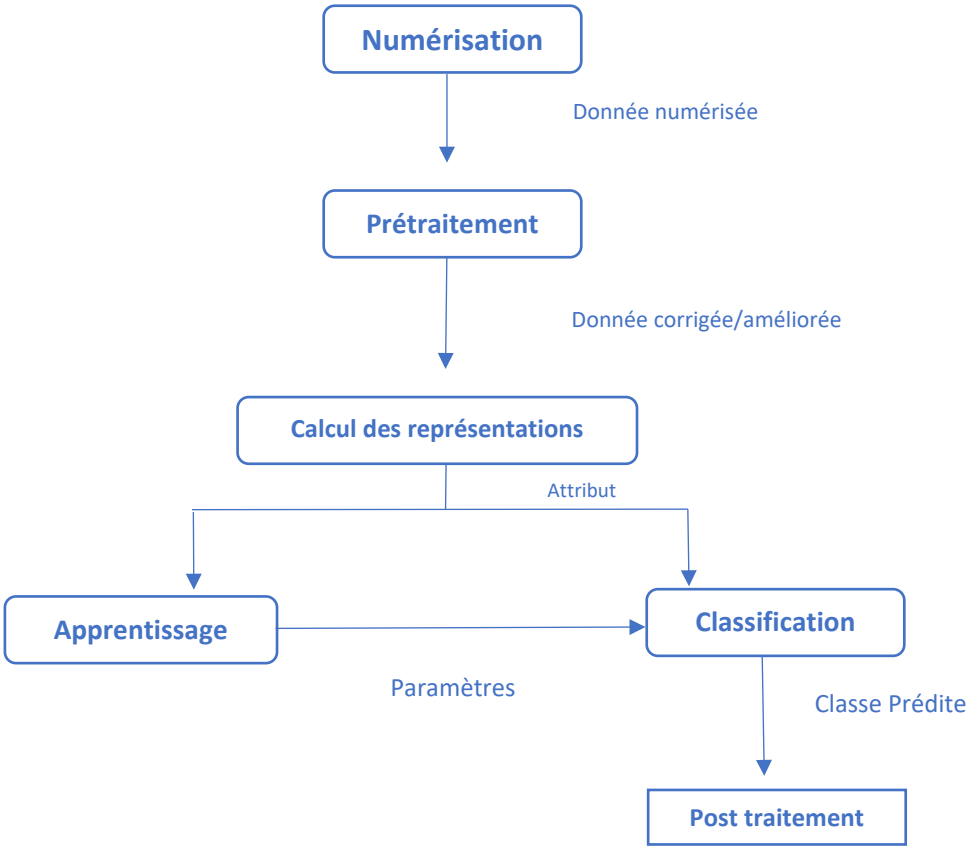


Figure 2. 2. Schéma général d'un système de reconnaissance des formes.

2.6.1 Préparation des données

2.6.1.1 Numérisation

À partir des informations du monde physique, construire une représentation des données directement manipulable par la machine. Des capteurs (microphone, caméra, instruments de mesure) convertissent les signaux reçus du monde réel en une représentation numérique discrète. L'espace résultant, appelé *espace de représentation* a une dimension **r** très grande lui permettant de disposer du maximum d'informations sur les formes numérisées [39].

2.6.1.2 Prétraitement

Consiste à sélectionner dans l'espace de représentation l'information nécessaire au domaine d'application. Cette sélection passe souvent par l'élimination du bruit, la normalisation des données, ainsi que par la suppression de la redondance. Le nouvel espace de représentation a une dimension **r'** très inférieure à **r** mais demeure un espace de grande dimension et contient des informations encore assez primitives [39].

2.6.1.3 Calcul des représentations

Il s'agit de la phase finale de la préparation des données. Elle fournit un certain nombre de caractéristiques ou paramètres (les fameux attributs) en utilisant des algorithmes de sélection et/ou d'extraction d'attributs. Les attributs étant limités en nombre, *l'espace des paramètres* ainsi obtenu est de dimension p très petite par rapport à **r'** [39].

2.6.2 Apprentissage

L'apprentissage ou entraînement, est une partie importante du système de reconnaissance. Le classificateur étant généralement une fonction paramétrique, l'apprentissage va permettre d'optimiser les paramètres du classificateur pour le problème à résoudre, en utilisant des données d'entraînement. Lorsque les données d'entraînement sont préalablement classées, l'apprentissage est dit supervisé, sinon il est non supervisé [37] [39].

2.6.3 Classification

Cette phase est le noyau de la Reconnaissance des formes. En utilisant les modèles (paramètres) obtenus lors de l'apprentissage, le classificateur assigne à chaque forme inconnue sa ou ses formes les plus probables [39].

2.6.4 Post traitement

Cette phase a pour but de corriger les résultats de la classification en utilisant des outils spécifiques au domaine d'application. Par exemple pour un système de reconnaissance de textes dans une image, le classificateur se charge de classer chaque caractère séparément, alors que le post traitement appliqué un correcteur orthographique sur tout le texte pour valider et éventuellement corriger le résultat de la classification. Bien que facultative, cette phase permet d'améliorer considérablement la qualité de la reconnaissance [39].

2.7 La vidéo

La vidéo regroupe l'ensemble des techniques, technologie, permettant l'enregistrement ainsi que la restitution d'images animées, accompagnées ou non de son, sur un support adapté à l'électronique et non de type photochimique. Le mot vidéo vient du latin vidéo qui signifie « je vois ». C'est l'apocope de vidéophonie ou vidéogramme.

Un flux vidéo est composé d'une succession d'images, 25 par seconde en Europe (30 par seconde aux USA), composant l'illusion du mouvement. Chaque image est décomposée en lignes horizontales, chaque ligne pouvant être considérée comme une succession de points. La lecture et la restitution d'une image s'effectue donc séquentiellement ligne par ligne comme un texte écrit : de gauche à droite puis de haut en bas.

2.7.1 Caractéristiques de la vidéo

Les principales caractéristiques de la vidéo sont : son nombre d'images par second, son entrelacement, sa résolution d'affichage, son allongement, la couleur espace et de bits par pixel, sa qualité et son début.

Nombre d'images par seconde

Le nombre d'images fixes par unité de temps de la vidéo, le nombre d'images par seconde fut normalisé à 25 images par seconde.

2.8 Traitement d'image

L'image fournie par le capteur est transformée en un signal électrique. De ce signal il faut extraire les informations recherchées sur le contenu de la scène dont l'image a été captée. Les premières bases du traitement d'images sont directement issues du traitement du signal, phénomène normal puisque toute image, qu'elle soit continue ou numérique, peut être considérée comme un signal à 2 dimensions [39].

La vision n'est pas un domaine facile, car repérer un objet simple dans une image demande beaucoup d'opérations. Le traitement, souvent appelé prétraitement, regroupe toutes les techniques visant à améliorer la qualité d'une image. De ce fait, la donnée de départ est l'image initiale et le résultat est également une image [39].

La représentation la plus élémentaire correspond à l'image binaire pour laquelle chaque pixel ne peut prendre qu'une valeur parmi deux autres. Pour les images monochromes, chaque

pixel peut prendre une valeur parmi N. N correspond généralement à une puissance de 2, ce qui facilite la représentation de l'image en machine. Par exemple, pour une image en niveau de gris chacun des pixels peut prendre une valeur parmi 256. Sa valeur est alors codée par un octet de donnée. Une image est constituée par une matrice X lignes et Y colonnes de pixels chacun codé par x bits [39]. Les données quantitatives liées à la représentation des images sont représentées dans le tableau suivant (Tableau 2.1) :

Tableau 2. 1. Données quantitatives des images.

1 bit	2 couleurs (noir & blanc)
4 bits	16 couleurs
8 bits	256 couleurs ou niveaux de gris
16 bits	65536 couleurs
24 bits	16 777 216 couleurs (vraies couleurs)
32 bits	4 294 967 296 couleurs

2.8.1 Définition de l'image

Une image est constituée d'un ensemble de points pixels (Picture Element). Il représente le plus petit élément constituant d'une image numérique (on parle d'image numérique lorsque les quantités physiques qui caractérisent l'image sont converties par des valeurs numériques). L'ensemble de ces pixels est contenu dans un tableau à deux dimensions constituant l'image. Les axes de l'image sont orientés de la façon suivante [39] :

- L'axe X est orienté à droite (largeur).
- L'axe Y est orienté de haut en bas (hauteur), contrairement aux notations conventionnelles en mathématiques, ou l'axe Y est orienté vers le haut.

Pour représenter informatiquement une image, il suffit donc de créer un tableau de pixels dont chaque case est codée par un certain nombre de débits déterminant la couleur ou l'intensité du pixel, la figure 2.4 présente un réseau de pixels (L'élément grisé encadré par la couleur rouge correspond aux coordonnées i, j) [39].

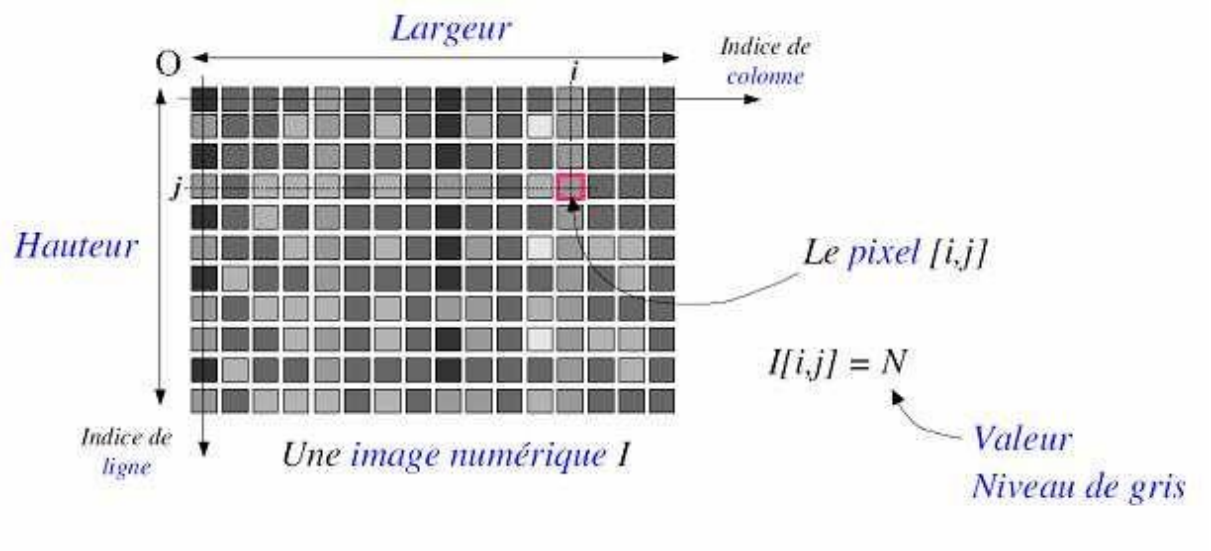


Figure 2. 3. Exemple de réseau de pixels [39].

Pixel : Le terme pixel est la contraction de l'anglais « Picture Element ». C'est le plus petit élément de l'image. Sortie de l'appareil photo une image est composée d'un nombre x de pixels. Un pixel a une couleur exprimée (codée) en langage binaire mathématique.

2.8.2 Acquisition d'une image

L'acquisition d'images constitue un des maillons essentiels de toute chaîne de conception et de production d'images. Pour pouvoir manipuler une image sur un système informatique, il est avant tout nécessaire de lui faire subir une transformation qui la rendra lisible et manipulable par ce système. Le passage de cet objet externe (l'image d'origine) à sa représentation interne (dans l'unité de traitement) se fait grâce à une procédure de numérisation. Ces systèmes de saisie, dénommés optiques, peuvent être classés en deux catégories principales :

- Les caméras numériques,
- Et les scanners.

A ce niveau, notons que le principe utilisé par le scanner est de plus en plus adapté aux domaines professionnels utilisant le traitement de l'image comme la télédétection, les arts graphiques, la médecine, etc. Le développement technologique a permis l'apparition de nouveaux périphériques d'acquisition appelés cartes d'acquisition, qui fonctionnent à l'instar des caméras vidéo, grâce à un capteur C.C.D. (Charge Coupled Device). La carte d'acquisition reçoit les images de la caméra, de la T.V. ou du scanner afin de les convertir en informations binaires qui seront stockées dans un fichier [39].

2.8.3 Caractéristiques d'une image numérique

2.8.3.1 Dimension

C'est la taille de l'image. Cette dernière se présente sous forme de matrice dont les éléments sont des valeurs numériques représentatives des intensités lumineuses (pixels). Le nombre de lignes de cette matrice multipliée par le nombre de colonnes nous donne le nombre total de pixels dans une image [39].

2.8.3.2 Résolution

C'est la clarté ou la finesse de détails atteinte par un moniteur ou une imprimante dans la production d'images. Sur les moniteurs d'ordinateurs, la résolution est exprimée en nombre de pixels par unité de mesure (pouce ou centimètre). On utilise aussi le mot résolution pour désigner le nombre total de pixels affichables horizontalement ou verticalement sur un moniteur ; plus grand est ce nombre, meilleure est la résolution [39].

2.8.3.3 Bruit

Un bruit dans une image est considéré comme un phénomène de brusque variation de l'intensité d'un pixel par rapport à ses voisins, il provient de l'éclairage des dispositifs optiques et électroniques du capteur [39].

2.8.3.4 Histogramme

L'histogramme des niveaux de gris ou des couleurs d'une image est une fonction qui donne la fréquence d'apparition de chaque niveau de gris (couleur) dans l'image. Il permet de donner un grand nombre d'information sur la distribution des niveaux de gris (couleur) et de voir entre quelles bornes est repartie la majorité des niveaux de gris (couleur) dans le cas d'une image trop claire ou d'une image trop foncée.

Il peut être utilisé pour améliorer la qualité d'une image (Rehaussement d'image) en introduisant quelques modifications, pour pouvoir extraire les informations utiles de celle-ci.
Pour diminuer l'erreur de quantification, pour comparer deux images obtenues sous des éclairages différents, ou encore pour mesurer certaines propriétés sur une image, on modifie souvent l'histogramme correspondant [39].

2.8.3.5 Luminance

C'est le degré de luminosité des points de l'image. Elle est définie aussi comme étant le quotient de l'intensité lumineuse d'une surface par l'aire apparente de cette surface, pour un observateur lointain, le mot luminance est substitué au mot brillance, qui correspond à l'éclat d'un objet. Une bonne luminance se caractérise par :

- Des images lumineuses (brillantes) ;
- Un bon contraste : il faut éviter les images où la gamme de contraste tend vers le blanc ou le noir ; ces images entraînent des pertes de détails dans les zones sombres ou lumineuses.
- L'absence de parasites.

2.8.3.6 Contraste

C'est l'opposition marquée entre deux régions d'une image, plus précisément entre les régions sombres et les régions claires de cette image. Le contraste est défini en fonction des luminances de deux zones d'images. Si L1 et L2 sont les degrés de luminosité respectivement de deux zones voisines A1 et A2 d'une image, le contraste C est défini par le rapport :

$$C = \frac{L_1 + L_2}{L_1 + L_2}$$

2.8.3.7 Images à niveaux de gris

Le niveau de gris est la valeur de l'intensité lumineuse en un point. La couleur du pixel peut prendre des valeurs allant du noir au blanc en passant par un nombre fini de niveaux intermédiaires. Donc pour représenter les images à niveaux de gris, on peut attribuer à chaque pixel de l'image une valeur correspondant à la quantité de lumière renvoyée. Cette valeur peut être comprise par exemple entre 0 et 255. Chaque pixel n'est donc plus représenté par un bit, mais par un octet. Pour cela, il faut que le matériel utilisé pour afficher l'image soit capable de produire les différents niveaux de gris correspondant. Le nombre de niveaux de gris dépend du nombre de bits utilisés pour décrire la " couleur " de chaque pixel de l'image. Plus ce nombre est important, plus les niveaux possibles sont nombreux.

Il existe plusieurs manières de convertir une image RGB en niveaux de gris. La plus simple est de faire :

$$gris = \frac{rouge + vert + bleu}{3}$$

Cela équivaut aussi à affecter la couleur en niveau de gris à chacune des trois composantes RGB. L'idéal est de faire ressortir la luminosité d'un pixel. Celle-ci vient principalement de la présence de la couleur verte. On emploie généralement les coefficients suivants [39] :

$$\text{Gris} = 0,299 \cdot \text{rouge} + 0,587 \cdot \text{vert} + 0,114 \cdot \text{bleu}$$

Dans ce cas on dispose d'une échelle de teintes de gris, et la plupart du temps on dispose de 256 niveaux de gris avec [68] :

0 ----> noir, ...127 ---> gris moyen, ..., 255 ---> blanc

Certaines images peuvent être codées sur deux octets ou plus (certaines images médicales, des images astronomiques...) ce qui peut poser des problèmes dans la mesure où les systèmes de traitement d'images courants supposent l'utilisation des pixels d'un octet.

2.8.3.8 Images en couleurs

Même s'il est parfois utile de pouvoir représenter des images en noir et blanc, les applications multimédias utilisent le plus souvent des images en couleurs. La représentation des couleurs s'effectue de la même manière que les images monochromes avec cependant quelques particularités. En effet, il faut tout d'abord choisir un modèle de représentation. On peut représenter les couleurs à l'aide de leurs composantes primaires. Les systèmes émettant de la lumière (écrans d'ordinateurs, ...etc.) sont basés sur le principe de la synthèse additive : les couleurs sont composées d'un mélange de rouge, vert et bleu (modèle R.V.B.).

2.8.4 Système de traitement d'images

Un système de traitement numérique d'images est composé de [39] :

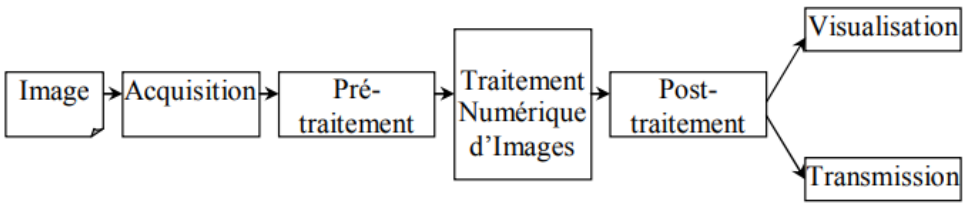


Figure 2. 4. Schéma d'un système de traitement d'images.

2.9 Segmentation des images

La segmentation est une étape essentielle en traitement d'images et reste un problème complexe. La segmentation est un processus de la vision par ordinateur, généralement c'est la première étape de l'analyse d'image qui vient après le prétraitement. La segmentation est l'extraction de caractéristiques de l'objet, ce qui permet une distinction entre l'objet et le fond [68]. La segmentation aide à localiser et à délimiter les entités présentes dans l'image. Il existe une multitude de méthodes de segmentation dont l'efficacité reste difficile à évaluer. La segmentation des images est le procédé qui conduit à un découpage de l'image en un nombre fini de régions (ou segments) bien définies qui correspondent à des objets, des parties d'objets ou des groupes d'objets qui apparaissent dans une image. C'est une transformation très utile en vision artificielle [39].

Une erreur dans la segmentation de la forme à reconnaître augmente forcément le risque d'une mauvaise reconnaissance. Essentiellement, l'analyse de l'image fait appel à la segmentation où l'on va tenter d'associer à chaque pixel de l'image un label en s'appuyant sur l'information portée (niveaux de gris ou couleur), sa distribution spatiale sur le support image, des modèles simples (le plus souvent des modèles géométriques) [39].

2.9.1 Types de Segmentation

2.9.1.1 Segmentation de texte en lignes

En général, le sous-titres ou le texte graphique ce compose de plusieurs lignes. En reconnaissance de texte, le texte se segmentant en lignes séparées. Il existe certaines méthodes utilisées à cet effet, telles que la projection horizontale.

2.9.1.2 Segmentation de lignes en caractères

Il s'agit ici de la segmentation de lignes en caractères individuels. Les points de segmentation sont identifiés à la fin d'un caractère et au début de la suivante.

2.9.2 Les principes de la segmentation

De nombreux travaux ont été réalisés sur ce sujet, dans des domaines aussi variés que le domaine médical, militaire, industriel, géophysique, etc… C'est toujours un sujet d'actualité et un problème qui reste ouvert où l'on retrouve de très nombreuses approches [68] :

- La segmentation par régions ;
- La segmentation par seuillage ;
- La segmentation par contours ;
- La Transformée de Hough ;
- La segmentation par étiquetage en composantes connexes ;
- La segmentation par LPE.

Toutes ces approches visent à l'extraction des caractéristiques. Après de nombreuses années passées à rechercher la méthode optimale, les chercheurs ont compris que la segmentation idéale n'existait pas. Il n'existe pas d'algorithme universel de segmentation à chaque type d'images correspond une approche spécifique. Une bonne méthode de segmentation sera donc celle qui permettra d'arriver à une bonne interprétation [39].

2.10 Reconnaissance de caractères de sous-titres vidéo

La reconnaissance de caractères de sous-titres vidéo est un traitement informatique qui a pour but de l'indexation vidéo basée sur le contenu, l'analyse de la vidéo et la traduction de sous-titres à d'autres langues.

2.10.1 Travaux connexes

Cette partie est consacrée pour l'analyse des résultats de différents articles qui s'intéressant sur la reconnaissance de caractères de sous-titres vidéo qui utilisent différentes méthodes de l'apprentissage automatique.

2.10.1.1 Le travail de Satish S. Hiremath et K.V. Suresh

Dans la recherche de M. Satish et M. Suresh [41] le système réalisé propose une reconnaissance des caractères en utilisant le classifieur support vecteur machine (SVM) pour la classification. Pour la validation du système, 23 classes de caractère ont été utilisées ce qui donne un taux de reconnaissance différent pour chaque caractère de chaque classe, Le nombre de caractères correctement reconnus par SVM est de 21 sur 23 soit 91%.

2.10.1.2 Le travail de Dorai et al

Dans le travail de Dorai et al [42] proposent une reconnaissance des caractères en utilisant le classifieur support vecteur machine (SVM) pour la classification. Dans leur travail ils utilisent une base de données de caractères contient 26 classes, ils ont testé des plusieurs configurations de SVM et le meilleur résultat est de 81,18%.

2.10.1.3 Le travail de Khaoula. E et Christophe. G

Dans le travail Khaoula. E et Christophe. G [43] utilisent une méthode de réseaux de neurones convolutionnel (CNN) pour la reconnaissance de caractères. Dans l'expérimentation ils utilisent une base de données de 15 168 images de caractères individuels parfaitement séparés, utilisée pour entrainer le réseau de neurones. 41 classes sont considérés : les 26 lettres, les 10 chiffres, 4 caractères spéciaux ('.','-', ' (', et ')'). Ils ont testé des architectures différentes de la méthode de CNN et le meilleur résultat est de 98%.

2.10.2 Comparaison entres les travaux réalisés

Les travaux cités dans la partie précédents concernant le domaine de la reconnaissance de caractères de sous-titres vidéo utilisent des différentes bases de données, la comparaison entre eux est fait en fonction des résultats obtenus et le nombre d'échantillons utilisées. Le but est de nous aider à proposer une approche en appliquant des nouvelles techniques sur une base de données plus riches avec des nombres, des lettres (majuscules et minuscules) et les marques de ponctuation.

Le tableau suivant donne un résumé des résultats des recherches mentionnés :

Tableau 2. 2. Les résultats des travaux connexes.

Articles	**Ensembles de données**	**Modèles**	**Résultats**
Satish S. Hiremath et K.V. Suresh [41]	23 classes de lettres.	Le classifieur support vecteur machine (SVM)	91%
Dorai et al [42]	26 classes de lettres.	Le classifieur support vecteur machine (SVM)	81,18%
Khaoula. E et Christophe. G [43]	41 classes de (10 - nombres, 26 – lettres, 4- marques de ponctuation)	Réseaux de neurones convolutionnel	98%

2.11 Conclusion

La Reconnaissance de Formes met en œuvre plusieurs étapes : segmentation des objets (analyse d'images), extraction de caractéristiques (géométrie, invariants, …), classification (supervisée ou non, méthodes probabilistes, statistiques, …). Les applications sont très variées et nécessitent des connaissances expertes du domaine d'application. Les méthodes sont nombreuses mais les principes de base sont assez stables.

Dans le prochain chapitre, Nous expliquant la conception générale de notre système de reconnaissance de caractères, ainsi les approches utilisées et on montre les résultats obtenus.

Chapitre 03 : Conception et implémentation.

3.1 Introduction

Ce dernier chapitre est consacré à la conception et la mise en place de notre projet qui permettra d'identifier des sous-titre vidéo par reconnaissance de formes en utilisant deux méthodes appartiennent à la famille de techniques de Deep learning. Dans ce chapitre nous allons décrire également les différentes parties de notre système, les détails relatifs à chaque phase ainsi que leurs interactions sont présentés dans les sous-sections suivantes.

3.2 Conception générale de notre système

Généralement, le processus de reconnaissance de caractères de sous-titres vidéo comporte cinq étapes :

1- Extraction des images de la vidéo.
2- Prétraitement d'image.
3- Segmentation des caractères présents dans les sous-titres.
4- Extraction de caractéristiques à partir de caractères segmentés.
5- Classification des caractères.

Après reçu la vidéo à traiter en entrée, notre système extrait les images de la vidéo comme première étape ce que produit un ensemble des images constituant la vidéo. Ensuite ces images vont passer à l'étape de « pré-traitement » image par image ce que produit en sortie deux types d'image : images sans sous-titre et images avec sous-titre. Le premier type d'image ne va pas passer à l'étape suivant tant que le deuxième type vas passer par l'étape suivante « segmentation ». Dans cette étape l'image de sous-titre vas être décomposée en ligne et après en imagette de caractères. Ces dernières vont passer image par image à l'étape suivante « reconnaissance » ce que produit finalement un texte de caractère. Il doit noter que cette étape effectuer plusieurs tâches : la construction du classifieur qui est ensuite permet d'extraire les caractéristiques principales de l'image et reconnaitre la classe correcte du caractère en image(classification).

La figure 3.1 d'écrit les différentes parties de notre système. Les détails relatifs à chaque phase ainsi que leurs interactions sont présentés dans les sous-sections suivantes.

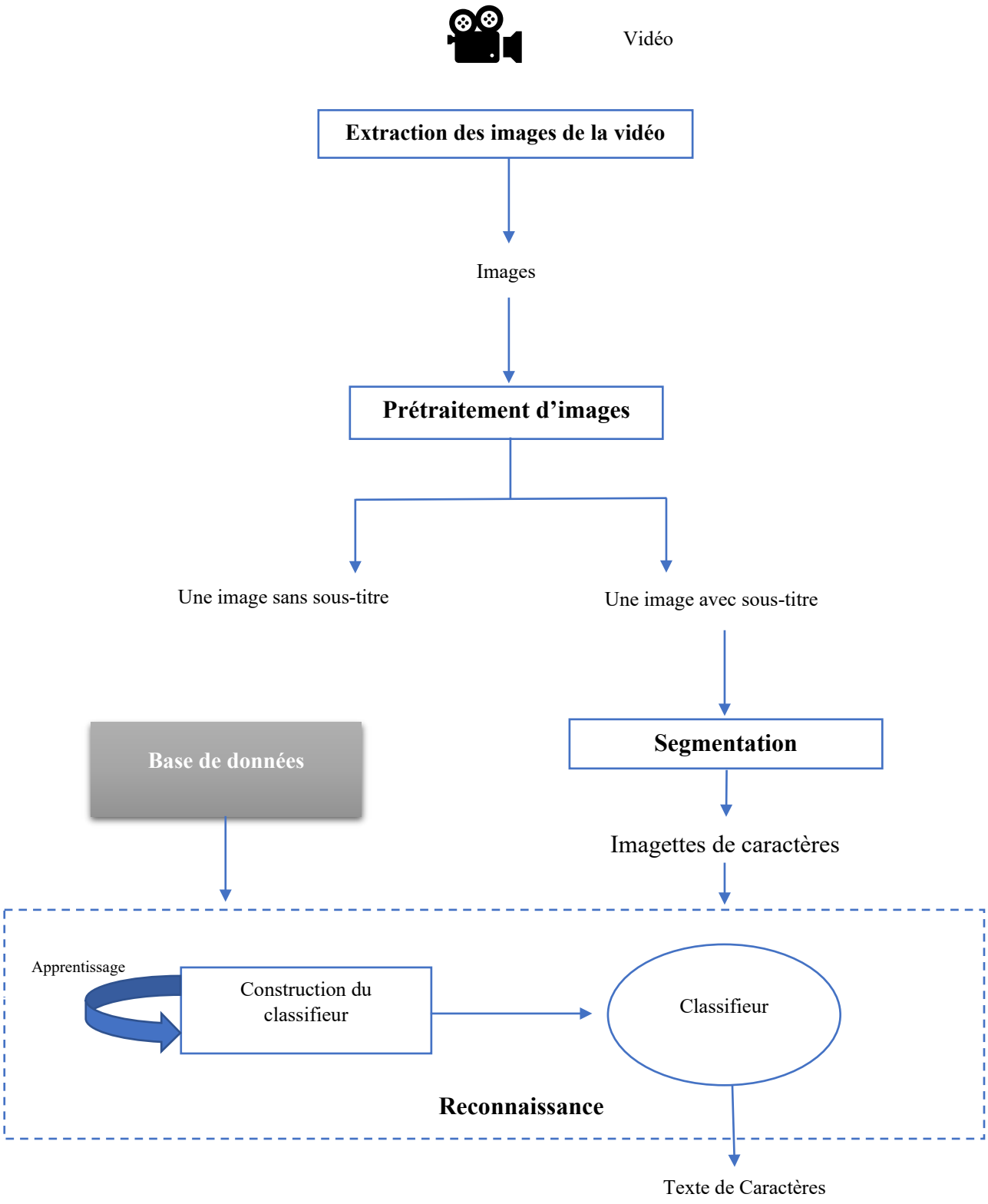


Figure 3. 1. Notre système de reconnaissance de caractères de sous titres vidéo.

3.3 Conception détaillé

3.3.1 Extraction des images de la vidéo

Au début de notre travail, Notre systèmes reçois comme entrée une vidéo. On sait que le flux vidéo est composé d'une succession d'images, 25 images par seconde. Chaque image est décomposée en lignes horizontales, chaque ligne pouvant être considérée comme une succession de points.

La première étape de notre système consiste à extraire les séquences de la vidéo, pour les extraire, nous avons appliqué une méthode consiste à faire une capture vidéo chaque une seconde.

A la fin de cette étape d'extraction, on obtient un dossier contient une multitude des images de la vidéo.

3.3.2 Prétraitement d'image

Cette étape est nécessaire dans notre système, chaque séquence de la vidéo va passer par trois opérations de prétraitement d'images qui sont :

1- Élimination de bruit
2- Sélectionne de la région du sous-titrage.
3- Vérification de l'existence de sous-titrage.

3.3.2.1 Élimination de bruit

Cette opération consiste à éliminer l'arriéré plan de la séquence vidéo pour facilite la détection de texte en images. La première étape de cette opération est la conversion de l'image originale en niveau de gris par le remplacement de chaque pixel de trois valeurs représentant les niveaux de rouge, de vert et de bleu, par une seule valeur représentant la luminosité. Puis, On applique le dé-bruitage par une fonction de seuillage. Finalement, On obtient une image avec texte claire en couleur blanc et l'arrière-plan noir.

3.3.2.2 Sélectionne de la région du sous-titre

Généralement dans le vidéo le sous-titrage est positionné en bas de l'image ou en haut. Cette opération consiste à vérifier si le sous-titrage est en bas en découpe la partie bas de l'image sinon la partie haut.

3.3.2.3 Vérification de l'existence de sous-titrage

Cette opération consiste à vérifier si l'image est avec sous-titre ou pas. On à utiliser une méthode basée sur la vérification de l'image si elle contient des pixels blancs donc avec sous-titrage ou elle est tous noir (sans sous-titrage).

A la fin de cette étape si notre image est avec sous-titrage elle doit passer par l'étape suivante qui est la segmentation, sinon on passe à l'image suivante.

3.3.3 Segmentation

Cette opération de segmentation consiste à décomposer, tout d'abord, le texte graphique (sous-titre) en lignes puis chaque ligne est segmentée en des caractères. Il existe plusieurs algorithmes de segmentation des images de textes, dans notre travail nous avons appliqué l'un des techniques de segmentation les plus récentes : algorithme de projection horizontale et verticale [45] (voir la figure 3.2). L'algorithme appliqué consiste à découper l'image de texte en des images de lignes et chaque image de ligne en imagettes de caractères.

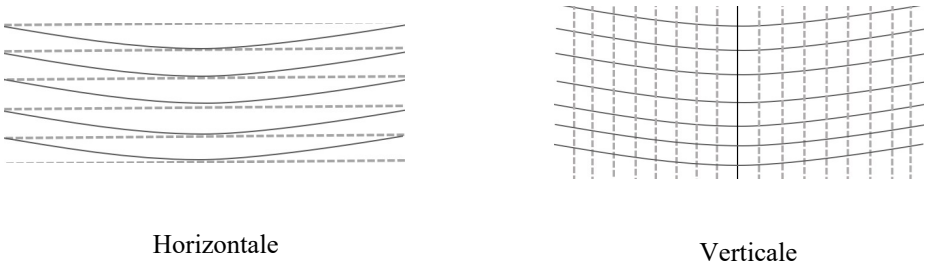


Figure 3. 2. La structure de découpage de l'image.

A l'issue de l'étape de segmentation, l'image originelle est décomposée en une multitude d'imagettes correspondant aux caractères. Le résultat de la segmentation de l'image vidéo avec sous-titres de la figure 3.3 est illustré sur la figure 3.4 et la figure 3.5**.**

Figure 3. 3. Une image avec sous-titres.

Figure 3. 4 le résultat de découpage de l'image avec sous-titres en deux images, chaque image contient une ligne de sous-titres.

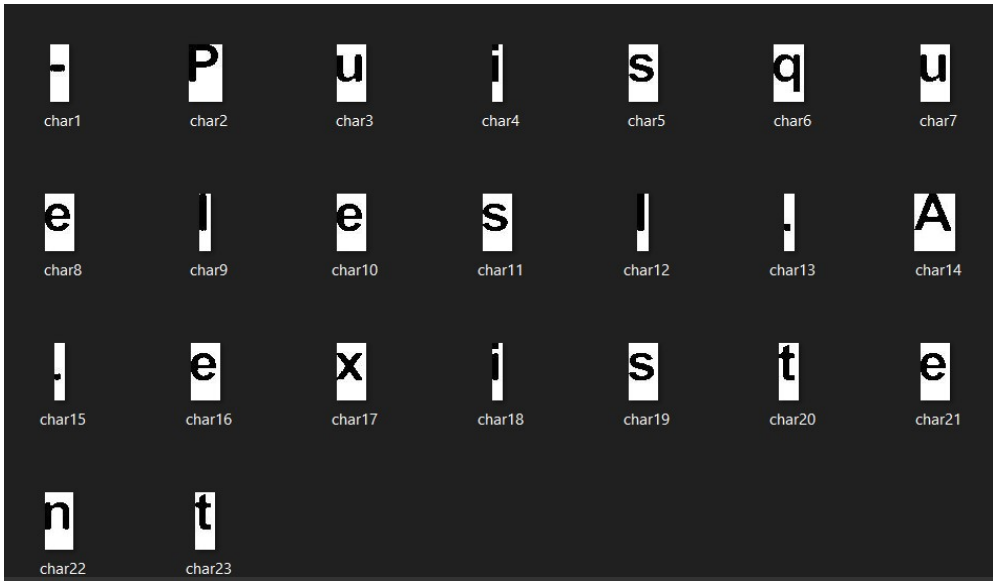


Figure 3. 5. Le résultat final de découpage d'une ligne d'un sous-titre en multitude d'imagettes de caractères.

3.3.4 Reconnaissance

Le processus de reconnaissance de caractères est le plus complexe. Généralement, il comporte deux phases : l'extraction de caractéristiques et la classification. L'extraction de caractéristiques permet de déterminer un vecteur dont les composantes caractérisent chaque type de caractère. La classification va permettre de déterminer la classe d'appartenance du caractère à l'aide de ce vecteur de caractéristiques.

Dans le présent travail deux approches sont utilisées et adopté pour atteindre les objectifs de cette phase : réseau neuronal convolutif (Convolutional neural network « CNN ») et Réseau neuronal à capsule (Capsule Neural Network « CapsNet »). Ces deux approches sont capables d'apprendre automatiquement et conjointement à extraire les primitives appropriées et à reconnaitre les classes de caractères, sans aucune phase de binarisation.

3.4 Système de reconnaissance à base de réseau neuronal convolutif

3.4.1 Architecture d'un CNN traditionnel

Les réseaux de neurones convolutionnels sont à ce jour permit les modèles les plus performants pour classer des images. Désignés par l'acronyme CNN, de l'anglais Convolutional Neural Network, ils comportent deux parties bien distinctes. En entrée, une image est fournie sous la forme d'une matrice de pixels. Elle a 2 dimensions pour une image en niveaux de gris. La couleur est représentée par une troisième dimension, de profondeur 3 pour représenter les couleurs fondamentales [Rouge, Vert, Bleu].

La première partie d'un CNN est la partie convolutive à proprement parler. Elle fonctionne comme un extracteur de caractéristiques des images. Une image est passée à travers une succession de filtres, ou noyaux de convolution, créant de nouvelles images appelées cartes de convolutions. Certains filtres intermédiaires réduisent la résolution de l'image par une opération de maximum local. Au final, les cartes de convolutions sont mises à plat et concaténées en un vecteur de caractéristiques, appelé code CNN [50].

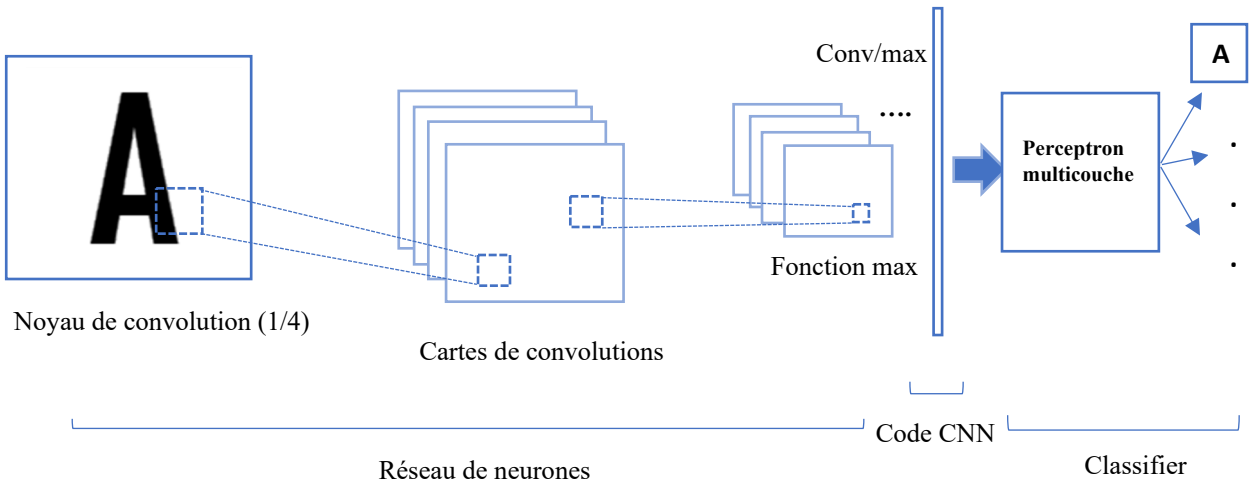


Figure 3. 6. Architecture standard d'un réseau de neurones convolutionnel.

Ce code CNN en sortie de la partie convolutive est ensuite branché en entrée d'une deuxième partie, constituée de couches entièrement connectées (perceptron multicouche). Le rôle de cette partie est de combiner les caractéristiques du code CNN pour classer l'image. La sortie est une dernière couche comportant un neurone par catégorie. Les valeurs numériques obtenues sont

généralement normalisées entre 0 et 1, de somme 1, pour produire une distribution de probabilité sur les catégories.

Les réseaux neuronales convolutifs sont généralement composés des couches suivantes (figure-3.7) :

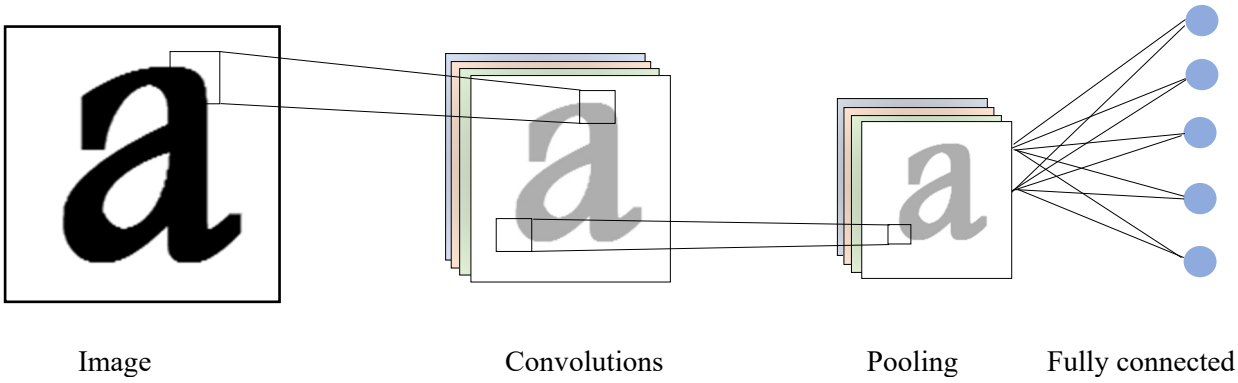


Figure 3. 7. L'architecture de CNN.

3.4.2 Les couches de réseaux de neurones convolutionnels

3.4.2.1 Couche convolutionnelle (CONV)

La couche convolutionnelle (en anglais convolution layer) (CONV) utilise des filtres qui scannent l'entrée **I** suivant ses dimensions en effectuant des opérations de convolution. Elle peut être réglée en ajustant la taille du filtre **F** et le stride **S**. La sortie **O** de cette opération est appelée *feature map* ou aussi *activation map* [49].

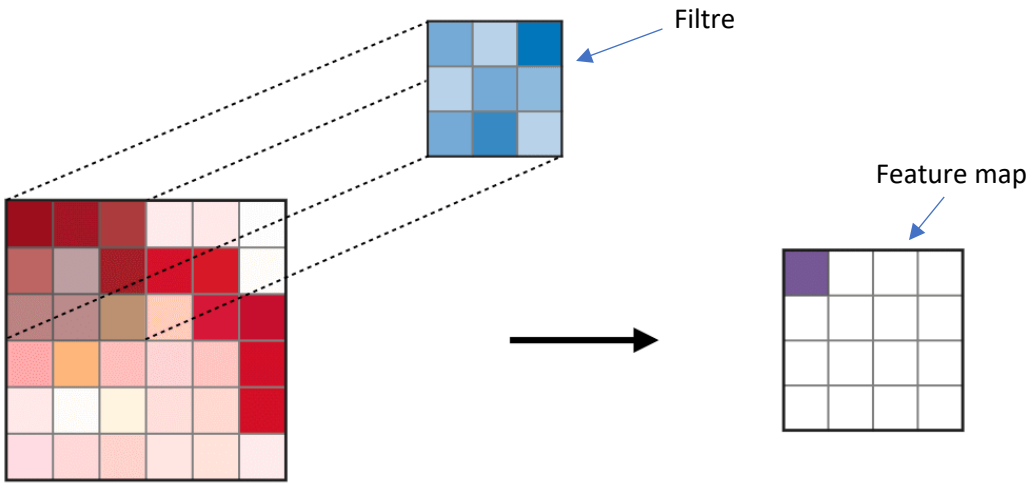


Figure 3. 8. La couche convolutionnelle.

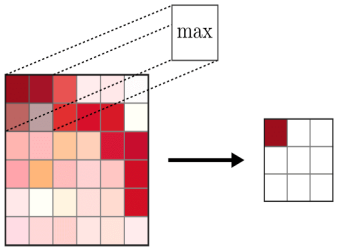
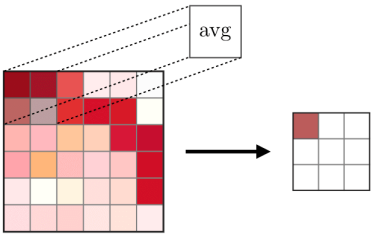
Remarque : l'étape de convolution peut aussi être généralisée dans les cas 1D et 3D

3.4.2.2 Couche de mise en commun (Pooling)

La couche de pooling (en anglais pooling layer « POOL ») est une opération de sous-échantillonnage typiquement appliquée après une couche convolutionnelle [49]. 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.

Le pooling fournir un gros gain 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. Les types de pooling les plus populaires sont le max et l'average pooling, où les valeurs maximales et moyennes sont prises, respectivement [49]. Le tableau ci-dessous illustre les deux types de la couche mise en commun.

Tableau 3. 1. Les types de la couche mise en commun.

Type	Max pooling	Average pooling
But	Chaque opération de pooling sélectionne la valeur maximale de la surface.	Chaque opération de pooling sélectionne la valeur moyenne de la surface.
Illustration		
Commentaires	- Garde les caractéristiques détectées. - Plus communément utilisé.	- Sous-échantillonne la feature map. - Utilisé dans LeNet.

3.4.2.3 Couche entièrement connecté (Fully connected)

La couche entièrement connecté (en anglais : Fully Connected Layer « FC ») s'applique sur une entrée préalablement aplatie où chaque entrée est connectée à tous les neurones. Les couches de *FC* sont typiquement présentes à la fin des architectures de CNN et peuvent être utilisées pour optimiser des objectifs tels que les scores de classe [49].

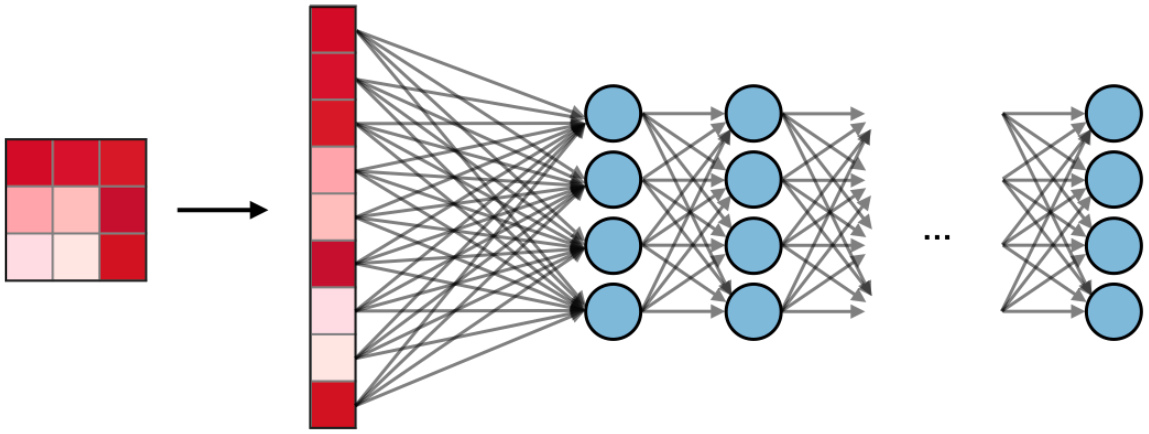


Figure 3. 9. La couche entièrement connectée.

3.4.3 Paramètres du filtre

La couche convolutionnelle (CONV) contient des filtres pour lesquels il est important de savoir comment ajuster ses paramètres.

3.4.3.1 Dimensions d'un filtre

Un filtre de taille **F × F** appliqué à une entrée contenant **C** canaux est un volume de taille **F × F × C** qui effectue des convolutions sur une entrée de taille **I × I × C** et qui produit un *feature map* de sortie (aussi appelé activation map) de taille **O × O × 1** [49].

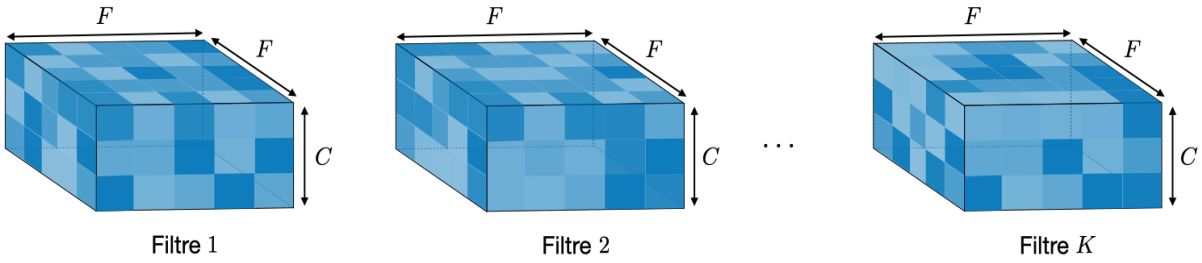


Figure 3. 10. Dimensions d'un filtre.

Remarque : appliquer K filtres de taille F × F engendre un *feature map* de sortie de taille O × O × K.

3.4.3.2 Stride

Dans le contexte d'une opération de convolution ou de *pooling*, la stride S est un paramètre qui dénote le nombre de pixels par lesquels la fenêtre se déplace après chaque opération [49].

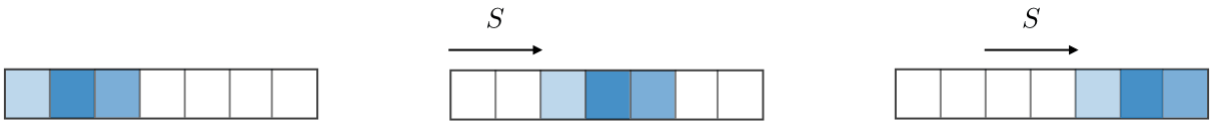
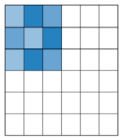
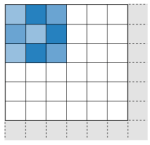
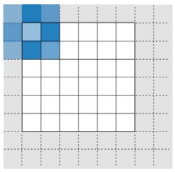


Figure 3. 11. La fenêtre se déplace avec 3 pixels (S=3) après chaque opération.

3.4.3.3 Zero-padding

Le zero-padding est une technique consistant à ajouter P zéros à chaque côté des frontières de l'entrée [49]. Cette valeur peut être spécifiée soit manuellement, soit automatiquement par le bias d'une des configurations détaillées dans le tableau ci-dessous :

Tableau 3. 2.les types de zero-padding.

Configuration	Valide	Pareil	Total
Valeur	$P = 0$	$P_{start} = \left\lfloor \frac{S\left\lceil \frac{I}{S} \right\rceil - I + F - S}{2} \right\rfloor$ $P_{end} = \left\lceil \frac{S\left\lceil \frac{I}{S} \right\rceil - I + F - S}{2} \right\rceil$	$P_{start} \in [\![0, F-1]\!]$ $P_{end} = F - 1$
Illustration			
But	- Pas de padding. - Enlève la dernière opération de convolution si les dimensions ne collent pas.	- Le padding tel que la *feature map* est de taille $\left\lceil \frac{I}{S} \right\rceil$. - La taille de sortie mathématiquement satisfaisante. - Aussi appelé 'demi' padding.	- Padding maximum tel que les dernières convolutions sont appliquées sur les bords de l'entrée. - Le filtre 'voit' l'entrée du début à la fin.

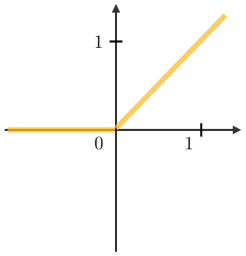
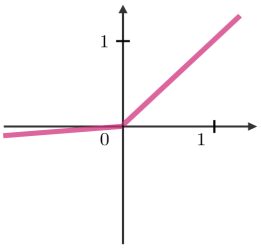
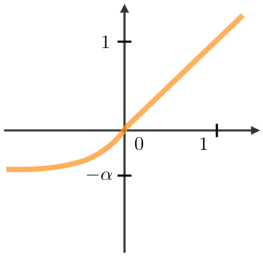
3.4.4 Fonctions d'activation

Les fonctions d'activation communément utilisées sont :

3.4.4.1 Unité linéaire rectifiée

La couche d'unité linéaire rectifiée (en anglais rectified linear unit layer « ReLU ») est une fonction d'activation **g** qui est utilisée sur tous les éléments du volume. Elle a pour but d'introduire des complexités non-linéaires au réseau [49] . Ses variantes sont récapitulées dans le tableau suivant :

Tableau 3. 3. Les variantes de la fonction d'activations.

ReLU	**Leaky ReLU**	**ELU**
$g(z) = \max(0, z)$	$g(z) = \max(\in z, z)$ Avec $\in \ll 1$	$g(z) = \max(\alpha(e^z - 1), z)$ Avec $\alpha \ll 1$
		
- Complexités non-linéaires interprétables d'un point de vue biologique.	- Répond au problème de Dying ReLU.	- Dérivable partout.

3.4.4.2 Softmax

L'étape softmax peut être vue comme une généralisation de la fonction logistique qui prend comme argument un vecteur de scores $x \in R^n$ et qui renvoie un vecteur de probabilités $p \in R^n$ à travers une fonction softmax à la fin de l'architecture [49].

Elle est définie de la manière suivante :

$$p = \begin{pmatrix} p_1 \\ . \\ . \\ . \\ p_n \end{pmatrix} \quad \text{Ou} \quad p_i = \frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}}$$

3.4.5 Pourquoi les réseaux de neurones convolutionnel (CNN) ?

Alors que les réseaux neuronaux et d'autres méthodes de détection de modèles existent depuis 50 ans, il y a eu un développement significatif dans le domaine des réseaux neuronaux convolutionnels dans un passé récent. Cette section présente les avantages de l'utilisation de CNN pour la reconnaissance d'images.

3.4.5.1 Robustesse des décalages et des déformations de l'image

La détection à l'aide de CNN est résistante aux distorsions telles que le changement de forme dû à l'objectif de la caméra, aux différentes conditions d'éclairage, différentes poses, présence d'occlusions partielles, décalages horizontaux et verticaux, etc. Cependant, les CNN sont invariants par décalage puisque la même configuration de poids est utilisée dans l'espace. En théorie, nous pouvons également atteindre l'invariantité de décalage en utilisant des couches entièrement connectées. Mais le résultat de l'entraînement dans ce cas est plusieurs unités avec des modèles de poids identiques à différents endroits de l'entrée. Pour apprendre ces configurations de poids, un grand nombre de séances d'entraînement seraient nécessaires pour couvrir l'espace de variations possibles [58].

3.4.5.2 Moins de besoins en mémoire

Dans ce même cas hypothétique où nous utilisons une couche entièrement connectée pour extraire les caractéristiques, l'image d'entrée de taille 32x32 et une couche cachée ayant 1000 caractéristiques nécessiteront un ordre de 106 coefficients, une énorme exigence de mémoire. Dans la couche convolutionnelle, les mêmes coefficients sont utilisés à

différents endroits dans l'espace, de sorte que l'exigence de mémoire est considérablement réduite [58].

3.4.5.3 Une formation plus facile et plus efficace

Encore une fois en utilisant le réseau neuronal standard qui serait équivalent à un CNN, parce que le nombre de paramètres serait beaucoup plus élevé, le temps de formation augmenterait également proportionnellement. Dans un CNN, le nombre de paramètres étant considérablement réduit, le temps de formation est proportionnellement réduit. En outre, en supposant une formation parfaite, nous pouvons concevoir un réseau neuronal standard dont la performance serait la même que celle d'un CNN. Mais dans la formation pratique, un réseau neuronal standard équivalent à CNN aurait plus de paramètres, ce qui conduirait à une augmentation du bruit pendant le processus de formation. Ainsi, la performance d'un réseau neuronal standard équivalent à un CNN sera toujours plus faible [58].

3.4.6 L'architecture de CNN proposé

Nous avons effectué plusieurs tests afin de retenir la meilleure architecture du CNN conduisant aux meilleurs taux de classification. Nous avons varié le nombre de couches, le nombre et la taille des filtres de convolution. Nous avons retenu le modèle de la figure 3.11. Le modèle ainsi développé est composé de six couches de convolution et trois couches de *Max-Pooling* et une couche de *FC*.

L'image en entrée est de taille 28x28, l'image passe d'abord à la première couche de convolution. Cette couche est composée de 32 filtres de taille 3x3, la fonction d'activation *ReLU* est utilisée pour forcer les neurones à retourner des valeurs positives. Cette convolution produit 32 *features maps* de taille 32x32 chacune. Ensuite, les 32 *feature maps* sont présentées en entrée à la deuxième couche de convolution qui est composée aussi de 32 filtres de taille 3x3. La fonction d'activation *ReLU* est appliquée sur les couches de convolutions. Le *Max-Pooling* est appliqué après pour réduire la taille de l'image et des paramètres. À la sortie de cette couche, nous obtenons 32 *featuremaps* de taille 16x16. On répète les mêmes opérations sur les couches de convolutions trois, quatre, cinq et six. Les couches trois et quatre sont composées de 64 filtres de tailles 3x3 et les couches cinq et six sont composées de 128 filtres de tailles 3x3. La fonction d'activation *ReLU* est appliquée toujours sur chaque convolution.

Une couche de *Max-Pooling* est appliquée après les couches de convolutions quatre et six. À la sortie de la dernière couche *Max-Pooling*, nous aurons 128 *featuremaps* de taille 2x2. Après ces six couches de convolution, nous utilisons un réseau de neurones composé d'une couche *FC*. Le nombre de neurones dans cette couche est égal au nombre de classes dans la base d'images. Les sorties des neurones de cette couche entièrement connectée sont converties en probabilités par l'intermédiaire d'une fonction d'activation ''softmax'.

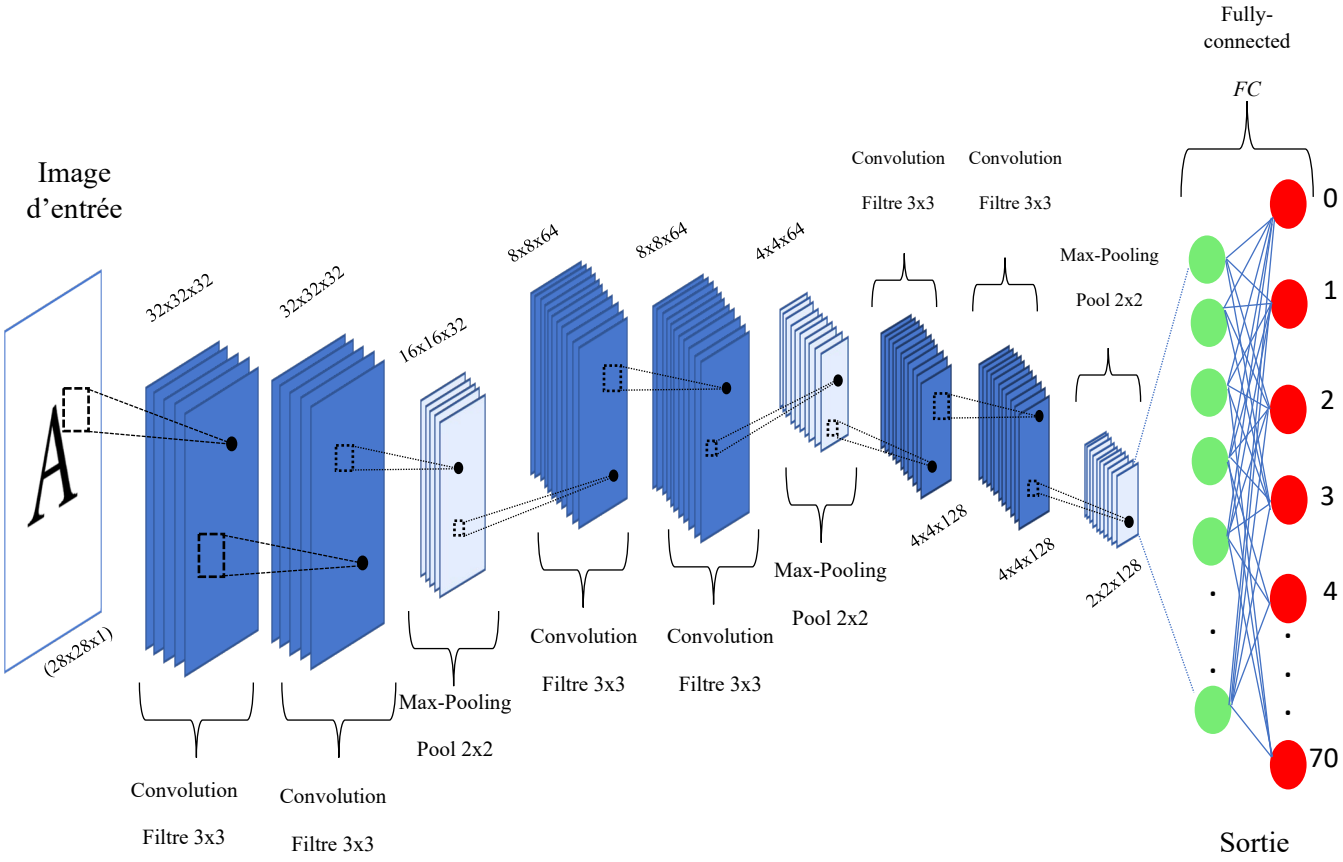


Figure 3. 12. L'architecture de modèle CNN utilisé dans notre système pour la reconnaissance de caractères.

3.5 Des astuces d'apprentissage profond

3.5.1 Traitement des données

3.5.1.1 Augmentation des données

Les modèles d'apprentissage profond ont typiquement besoin de beaucoup de données afin d'être entrainés convenablement. Il est souvent utile de générer plus de données à partir de celles déjà existantes à l'aide de techniques d'augmentation de données [46].

Celles les plus souvent utilisées sont résumées dans le tableau ci-dessous. À partir d'une image, voici les techniques que l'on peut utiliser :

Tableau 3. 4. Les techniques d'augmentations de données.

Original	**Symétrie axiale**	**Rotation**	**Recadrage aléatoire**
			
- Image sans aucune modification.	- Symétrie par rapport à un axe pour lequel le sens de l'image est conservé.	- Rotation avec un petit angle. - Reproduit une calibration imparfaite de l'horizon.	- Concentration aléatoire sur une partie de l'image. - Plusieurs rognements aléatoires peuvent être faits à la suite.

Changement de couleur	**Addition de bruit**	**Perte d'information**	**Changement de contraste**
			
- Nuances de RGB sont légèrement changées. - Capture le bruit qui peut survenir avec de l'exposition lumineuse.	- Addition de bruit. - Plus de tolérance envers la variation de la qualité de l'entrée.	- Parties de l'image ignorées. - Imite des pertes potentielles de parties de l'image.	- Changement de luminosité. - Contrôle la différence de l'exposition dû à l'heure de la journée.

3.5.1.2 Normalisation de lot

La normalisation de lot (en anglais batch normalization) est une étape qui normalise le lot $\{x_i\}$avec un choix de paramètres γ,β.

En notant $\mu_{\mathrm{B}}, \sigma_{\mathrm{B}}^2$ la moyenne et la variance de ce que l'on veut corriger du lot, on a :

$$x_i \leftarrow \gamma \frac{x_i - \mu_{\mathrm{B}}}{\sqrt{\sigma_{\mathrm{B}}^2 + \varepsilon}} + \beta$$

Ceci est couramment fait après la couche *FC* et avant la couche non-linéaire. Elle vise à permettre d'avoir de plus grands taux d'apprentissages et de réduire la dépendance à l'initialisation [46] .

3.5.2 Entraîner un réseau de neurones

3.5.2.1 Itération

Mise à jour unique des pondérations d'un modèle pendant l'apprentissage. Une itération consiste à calculer les gradients des paramètres en termes de perte sur un seul lot de données.

3.5.2.2 Lot (batch)

Ensemble d'exemples utilisés dans une itération (c'est-à-dire, une mise à jour du gradient) de l'entraînement du modèle.

3.5.2.3 Taille de lot (batch size)

Nombre d'exemples d'un lot. La taille de lot est habituellement fixée pendant les processus d'apprentissage.

3.5.2.4 Époque (Epoch)

Cycle d'apprentissage complet sur l'intégralité de l'ensemble de données de manière à ce que chaque exemple ait été vu une fois. Une itération représente ainsi *N/taille du lot* itérations d'apprentissage, où *N* est le nombre total d'exemples.

3.5.2.5 Gradient descent sur mini-lots

Durant la phase d'entraînement, la mise à jour des coefficients n'est souvent basée ni sur tout le training set d'un coup à cause de temps de calculs coûteux, ni sur un seul point à cause de bruits potentiels. À la place de cela, l'étape de mise à jour est faite sur des mini-lots, où le nombre de points dans un lot est un paramètre que l'on peut régler [46].

3.5.2.6 Fonction de loss

Pour pouvoir quantifier la performance d'un modèle donné, la fonction de loss (en anglais *loss function*) est utilisée pour évaluer la mesure dans laquelle les sorties vraies $\boldsymbol{y}$ sont correctement prédites par les prédictions du modèle $\boldsymbol{z}$ [46].

3.6 La base de données

Les réseaux de neurones ont besoin d'une grande quantité de données d'entrainement sur les images de caractères pour obtenir un bon résultat. Nous avons construit une base de données contenant plus de 49000 images de caractères (nombres, lettres majuscules et minuscules et les marques de ponctuation).la bases de données est divisé en 70 classes. Comme suit :

De 1 à 10 classes sont des nombres.

De 11 à 62 classes sont des lettres majuscules et minuscules [A-Z, a-z].

De 63 à 70 classes sont des marques de ponctuation. Tél que : ?, !, :, -, (,) …etc.

Cette base est divisée en deux parties une pour l'apprentissage contenant 40000 images et l'autre pour le test contenant 9848 images.

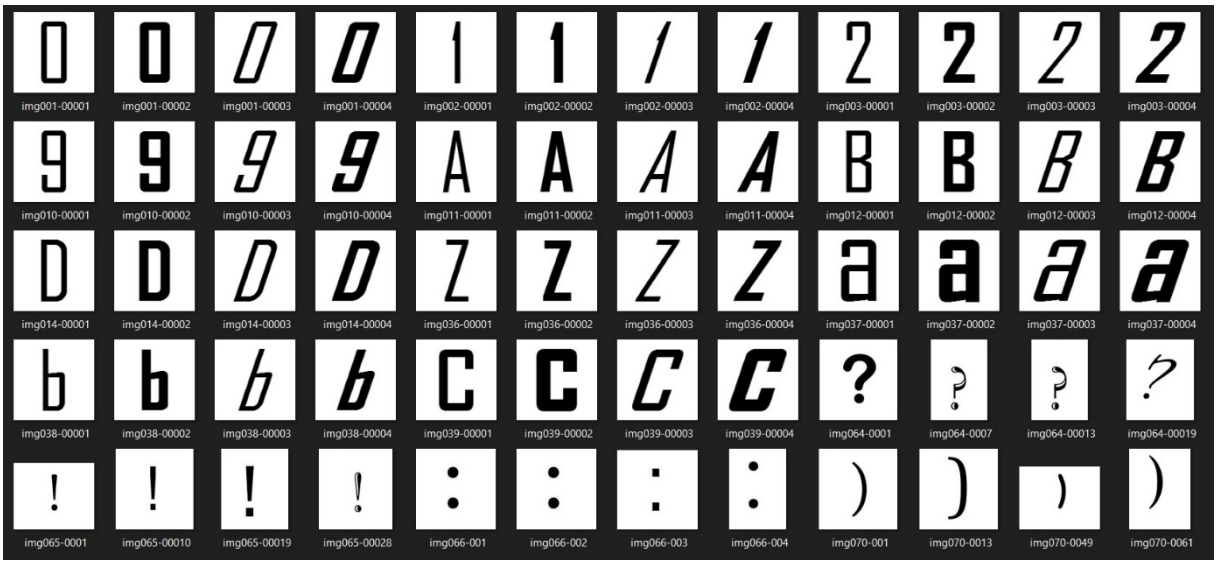


Figure 3. 13. Échantillons de la base de données.

3.7 Langage, Logiciels et librairies utilisés dans l'implémentation

3.7.1 Python

Python est un langage de programmation de haut niveau interprété (il n'y a pas d'étape de compilation) et orienté objet avec une sémantique dynamique. Il est très sollicité par une large communauté de développeurs et de programmeurs. Python est un langage simple, facile à apprendre et permet une bonne réduction du cout de la maintenance des codes. Les bibliothèques (packages) python encouragent la modularité et la réutilisabilité des codes. Python et ses bibliothèques sont disponibles (en source ou en binaires) sans charges pour la majorité des plateformes et peuvent être redistribués gratuitement [57].

3.7.2 Tensorflow

TensorFlow est une bibliothèque de logiciels open source pour le calcul numérique haute performance. Son architecture flexible permet un déploiement facile du calcul sur une variété de plates-formes (CPU, GPU, TPU), et des ordinateurs de bureau aux clusters de serveurs en passant par les périphériques mobiles et périphériques.

Développé à l'origine par des chercheurs et des ingénieurs de l'équipe Google Brain au sein de l'organisation IA de Google, il est livré avec un support solide pour l'apprentissage automatique et l'apprentissage en profondeur et le noyau de calcul numérique flexible est utilisé dans de nombreux autres domaines scientifiques [52].

3.7.3 Keras

Keras est une API de réseaux de neurones de haut niveau, écrite en Python et capable de fonctionner sur TensorFlow. Il a été développé en mettant l'accent sur l'expérimentation rapide. Être capable d'aller de l'idée à un résultat avec le moins de délai possible est la clé pour faire de bonnes recherches. Il a été développé dans le cadre de l'effort de recherche du projet ONEIROS (Open-ended Neuro-Electronic Intelligent Robot Operating System), et son principal auteur et mainteneur est François Chollet, un ingénieur Google.
En 2017, l'équipe TensorFlow de Google a décidé de soutenir Keras dans la bibliothèque principale de TensorFlow. Chollet a expliqué que Keras a été conçue comme une interface plutôt que comme un cadre d'apprentissage end to end. Il présente un ensemble d'abstractions de niveau supérieur et plus intuitif qui facilitent la configuration des réseaux neuronaux indépendamment de la bibliothèque informatique de backend. Microsoft travaille également à ajouter un backend CNTK à Keras aussi [53].

3.7.4 PIL

Pillow est une bibliothèque de traitement d'image, qui est un fork et successeur du projet PIL (*Python Imaging Library*). Elle est conçue de manière à offrir un accès rapide aux données contenues dans une image, et offre un support pour différents formats de fichiers tels que PPM, PNG, JPEG, GIF, TIFF et BMP.

Pillow dispose de capacités de traitement d'images relativement puissantes, et a pour but d'offrir une solide base à toute application générale de traitement d'images [51].

3.7.5 OpenCV

OpenCV (pour Open Computer Vision) est une bibliothèque graphique libre, initialement développée par Intel, spécialisée dans le traitement d'images en temps réel. La société de robotique Willow Garage et la société ItSeez se sont succédé au support de cette bibliothèque. Depuis 2016 et le rachat de ItSeez par Intel, le support est de nouveau assuré par Intel [56].

Cette bibliothèque est distribuée sous licence BSD.

3.7.6 Tkinter

Tkinter (de l'anglais Tool kit interface) est la bibliothèque graphique libre d'origine pour le langage Python, permettant la création d'interfaces graphiques. Elle vient d'une adaptation de la bibliothèque graphique Tk écrite pour Tcl [54].

3.7.7 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, et supporte le développement web avec Django.

Développé par l'entreprise tchèque JetBrains, c'est un logiciel multi-plateforme qui fonctionne sous Windows, Mac OS X et Linux [55].

3.7.8 Configuration utilisée dans l'implémentation

La configuration du matériel utilisé dans notre implémentation est :

- Un PC portable Lenovo i7 8 gén CPU 1.8 GHZ.
- Carte graphique Nvidia MX150 4GB.
- RAM de taille 8 GO.
- Disque dur de taille 2 T HDD.
- Système d'exploitation Windows 10.

3.8 Résultat obtenu pour le modèle de CNN

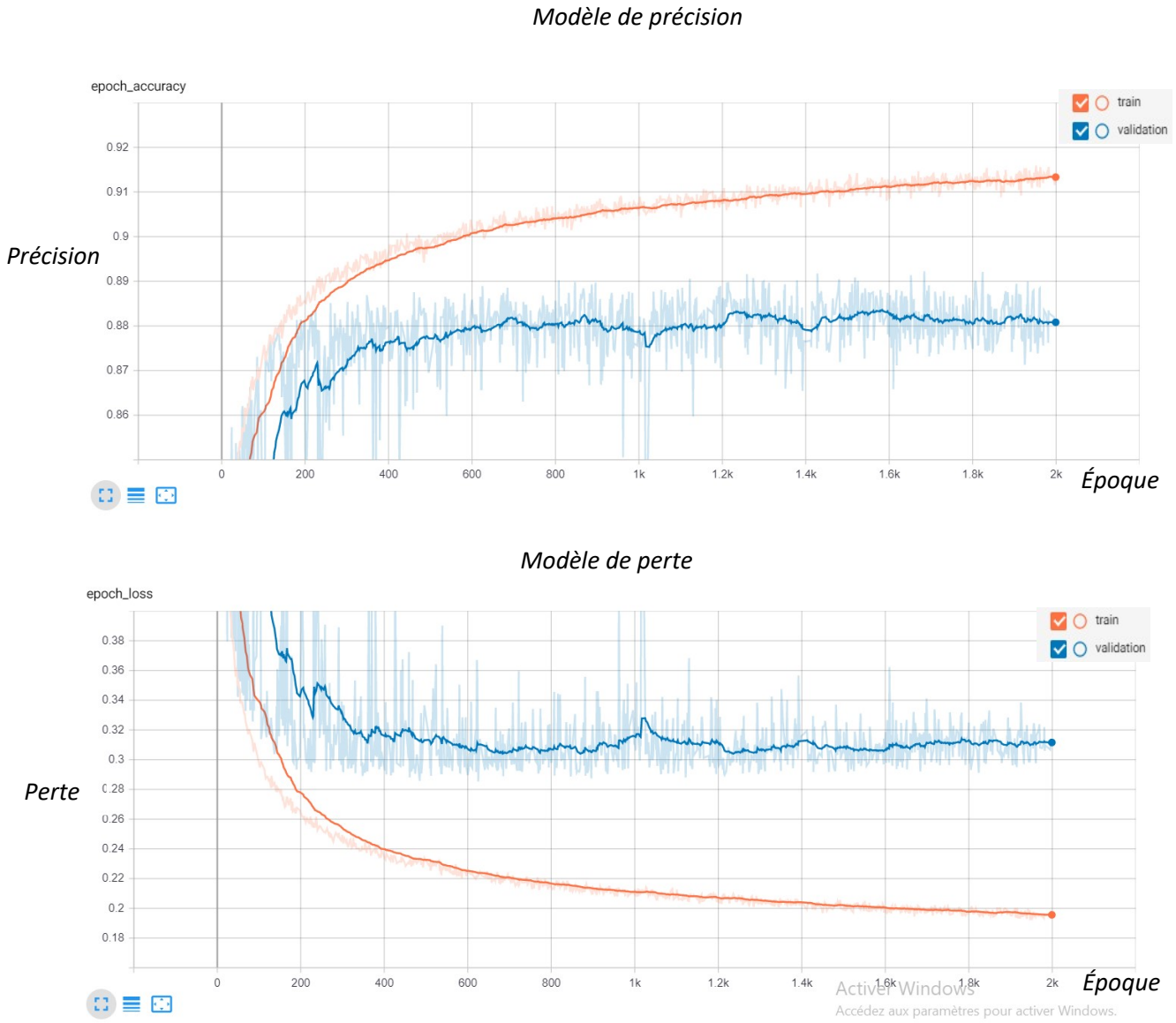


Figure 3. 14. La précision et l'erreur de modèle de CNN.

Après l'analyse des résultats obtenus, Nous avons constaté les remarques suivantes :

- D’après la Figure 3.14 La précision de l’apprentissage et de test augmente avec le nombre d’époque, ceci reflète qu’à chaque époque le modèle apprenne plus d’informations.
- Si la précision est diminuée alors on aura besoin de plus d’information pour faire apprendre notre modèle et par conséquent on doit augmenter le nombre d’époque.
- De même, l’erreur d’apprentissage et de la validation diminue avec le nombre d’époque.

3.8.1 Les limites de réseau de neuronal convolutionnel

Malgré que les CNN (réseaux de neurones convolutifs) sont géniaux, ils ont leur limites et ils présentent des inconvénients fondamentaux.

Prenons un exemple très simple et non technique. Imaginez un visage. Quels sont les composants d’un visage ? Nous avons le visage ovale, deux yeux, un nez et une bouche. Pour un CNN, la simple présence de ces objets peut être un indicateur très fort pour considérer qu'il y a un visage dans l'image. Les relations spatiales d'orientation et relatives entre ces composants ne sont pas très importantes pour un CNN. (Voir la figure 3.14)

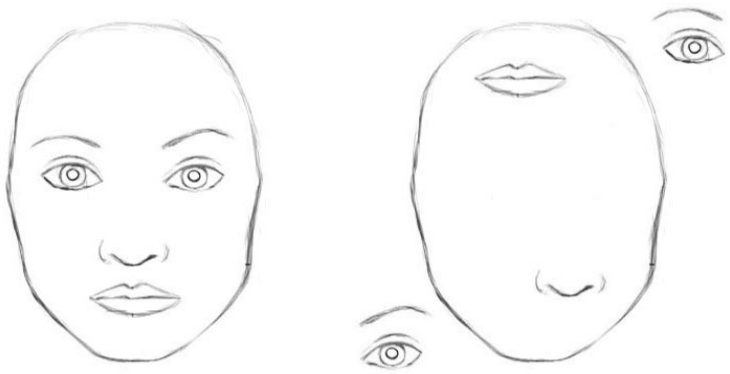


Figure 3. 15. Pour un CNN, les deux images (A et B) sont similaires, car elles contiennent toutes deux des éléments similaires.

La relation spatiale entre caractéristiques détectées par les phases successives de convolution est partiellement assurée par l’introduction de couches de neurones, les couches de *Pooling*, permettant de compresser l’information en réduisant la taille de l’image intermédiaire et ainsi d’élargir le champ de vision du réseau. Concrètement deux yeux et une bouche apparaissant

sur le même champ de vision grâce à l'opération de *Pooling* seront alors associés à un objet de plus haut niveau, un visage.

Pour balayés ces inconvénients, nous avons recrutés à l'utilisation d'une autre approche d'apprentissage profond (*DL*) qui est l'un des extensions de CNN qui permet d'évaluer les performances et de corriger les défauts de CNN qui est L'architecture des réseaux de capsules (en anglais << Capsule neural networks >>).

3.9 Système de reconnaissance à base de réseau neuronal à capsule (CapsNet)

Un CapsNet (ou réseau neuronal à capsule) est un système de machine learning de la famille des réseaux neuronaux artificiels (RNA) pouvant être utilisé pour mieux modéliser des relations hiérarchiques. Cette approche est une tentative de reproduire plus fidèlement une organisation neuronale biologique.

L'idée est d'ajouter des structures appelées « capsules » à un réseau de neurones à convolution (CNN) et de réutiliser les sorties de plusieurs de ces capsules pour former des représentations plus stables (par rapport à diverses perturbations) de capsules d'ordre supérieur. La sortie est un vecteur consistant en la probabilité d'une observation et une pose pour cette observation. Ce vecteur est similaire à ce qui est fait par exemple lors de la classification avec localisation dans CNN.

Un réseau neuronal à capsule permet de dépasser les limites des réseaux de neurones convolutifs utilisés jusqu'à aujourd'hui notamment pour la reconnaissance d'images, en dotant les réseaux de neurones d'une capacité à prendre en compte des informations contextuelles sur les caractéristiques extraites de l'image, notamment leur organisation spatiale et hiérarchique au sein de l'image.

3.9.1 Capsules

Les neurones artificiels produisent un seul scalaire. De plus, les CNN utilisent des couches convolutives qui, pour chaque noyau, répliquent les poids de ce même noyau sur tout le volume d'entrée, puis produisent une matrice 2D, où chaque nombre est la sortie de la convolution de ce noyau avec une partie du volume d'entrée. Nous pouvons donc regarder cette matrice 2D en tant que sortie du détecteur de caractéristiques répliquées. Ensuite, toutes les matrices 2D du noyau sont empilées les unes sur les autres pour produire la sortie d'une couche convolutionnelle [47].

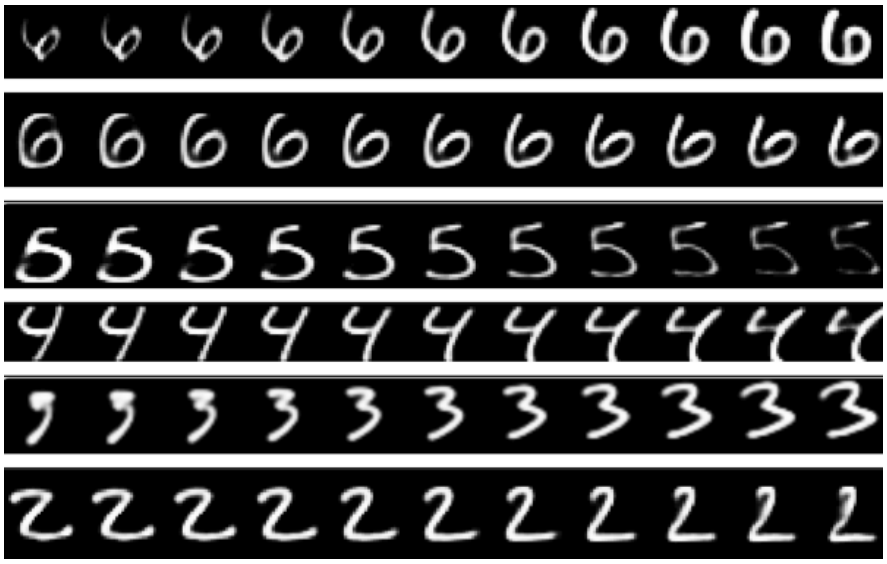


Figure 3. 16. Le CapsNet peut générer les chiffres à partir de représentions internes [44].

Ensuite, nous essayons d'obtenir une invariance des points de vue dans les activités des neurones. Nous faisons cela au moyen de la mise en commun maximale qui examine consécutivement les régions dans la matrice 2D décrite ci-dessus et sélectionne le plus grand nombre dans chaque région. En conséquence, nous obtenons ce que nous voulions : l'invariance des activités. L'invariance signifie qu'en modifiant un peu l'entrée, la sortie reste la même. Et l'activité n'est que le signal de sortie d'un neurone. En d'autres termes, lorsque dans l'image d'entrée, nous décalons un peu l'objet que nous voulons détecter, les activités du réseau (sorties des neurones) ne changeront pas en raison du pooling maximal et le réseau détectera toujours l'objet [47].

Le mécanisme décrit ci-dessus n'est pas très bon, car la mise en commun maximale perd des informations précieuses et n'encode pas non plus les relations spatiales relatives entre les entités. Nous devrions plutôt utiliser des capsules, car elles encapsuleront toutes les informations importantes sur l'état des caractéristiques qu'elles détectent sous la forme d'un vecteur (par opposition à un scalaire produit par un neurone). Les capsules encapsulent toutes les informations importantes sur l'état de la caractéristique qu'elles détectent sous forme vectorielle [44].

Les capsules codent la probabilité de détection d'une entité comme la longueur de leur vecteur de sortie. Et l'état de la caractéristique détectée est codé comme la direction dans laquelle ce vecteur pointe (paramètres d'instanciation). Ainsi, lorsque la fonction détectée se déplace autour de l'image ou que son état change d'une manière ou d'une autre, la probabilité reste la même (la longueur du vecteur ne change pas), mais son orientation change.

3.9.1.1 Fonctionnement d'une capsule

Comparons les capsules aux neurones artificiels. Le tableau ci-dessous résume les différences entre la capsule et le neurone [47] :

Tableau 3. 5. Les différences entre la capsule et le neurone.

		Capsule	**Neurone traditionnel**
Entrée de bas niveau		Vecteur ($\boldsymbol{u}_i$)	Scalaire ($\boldsymbol{x}_i$)
Opération	**Transformation affine**	$\hat{u}_{j\|i} = \boldsymbol{W}_{ij}\boldsymbol{u}_i$	—
	Pondération / Somme	$\boldsymbol{s}_j = \sum_i c_{ij}\hat{u}_{j\|i}$	$a_j = \sum_i W_i \boldsymbol{x}_i + b$
	Activation non linéaire	$\boldsymbol{v}_j = \frac{\\|\boldsymbol{s}_j\\|^2}{1+\\|\boldsymbol{s}_j\\|^2}\frac{\boldsymbol{s}_j}{\\|\boldsymbol{s}_j\\|}$	$h_i = f(a_j)$
Sortie		Vecteur ($\boldsymbol{v}_j$)	Scalaire (h_i)

On sait qu'un neurone reçoit des scalaires d'entrée d'autres neurones, puis les multiplie par des poids et des sommes scalaires. Cette somme est ensuite transmise à l'une des nombreuses fonctions d'activation non linéaires possibles, qui prennent le scalaire d'entrée et produisent un scalaire en fonction de la fonction. Ce scalaire sera la sortie du neurone qui ira comme entrée à

d'autres neurones. Le résumé de ce processus peut être vu sur le tableau et le diagramme ci-dessous (Figure 3.16) sur le côté droit [47]. En substance, le neurone artificiel peut être décrit en 3 étapes :

1. Pondération scalaire des scalaires d'entrée.
2. Somme des scalaires d'entrée pondérés.
3. Non-linéarité scalaire à scalaire.

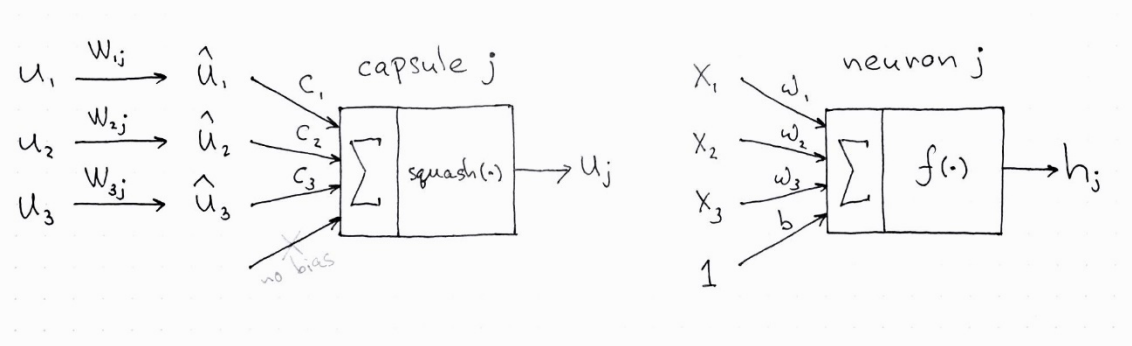


Figure 3. 17. Diagramme qui montre la différence entre les capsules et les neurones.

D'autre part, la capsule a des formes vectorielles des 3 étapes ci-dessus en plus de la nouvelle étape, transformée affine d'entrée :

1.Multiplication matricielle des vecteurs d'entrée.

2.Pondération scalaire des vecteurs d'entrée.

3.Somme des vecteurs d'entrée pondérés.

4. Non-linéarité de vecteur à vecteur.

3.9.1.2 Les 4 étapes de calcul qui se déroulent à l'intérieur de la capsule.

a) Multiplication matricielle des vecteurs d'entrée

Les vecteurs d'entrée que reçoit notre capsule (u_1, $u_2$et u_3 dans le diagramme) proviennent de 3 autres capsules de la couche inférieur. Les longueurs de ces vecteurs codent les probabilités que les capsules de niveau inférieur ont détecté leurs objets correspondants et les directions des vecteurs codent un état interne des objets détectés [47].

Les vecteurs sont ensuite multipliés par des matrices de poids $\boldsymbol{W}$ correspondantes qui codent des relations spatiales et autres importantes entre des caractéristiques de niveau inférieur et une caractéristique de niveau supérieur [47].

b) Pondération scalaire des vecteurs d'entrée

À première vue, cette étape semble très familière à celle où le neurone artificiel pondère ses entrées avant de les additionner. Dans le cas des neurones, ces poids sont appris lors de la rétropropagation, mais dans le cas de la capsule, ils sont déterminés en utilisant le « routage dynamique », qui est une nouvelle façon de déterminer où va la sortie de chaque capsule [47].

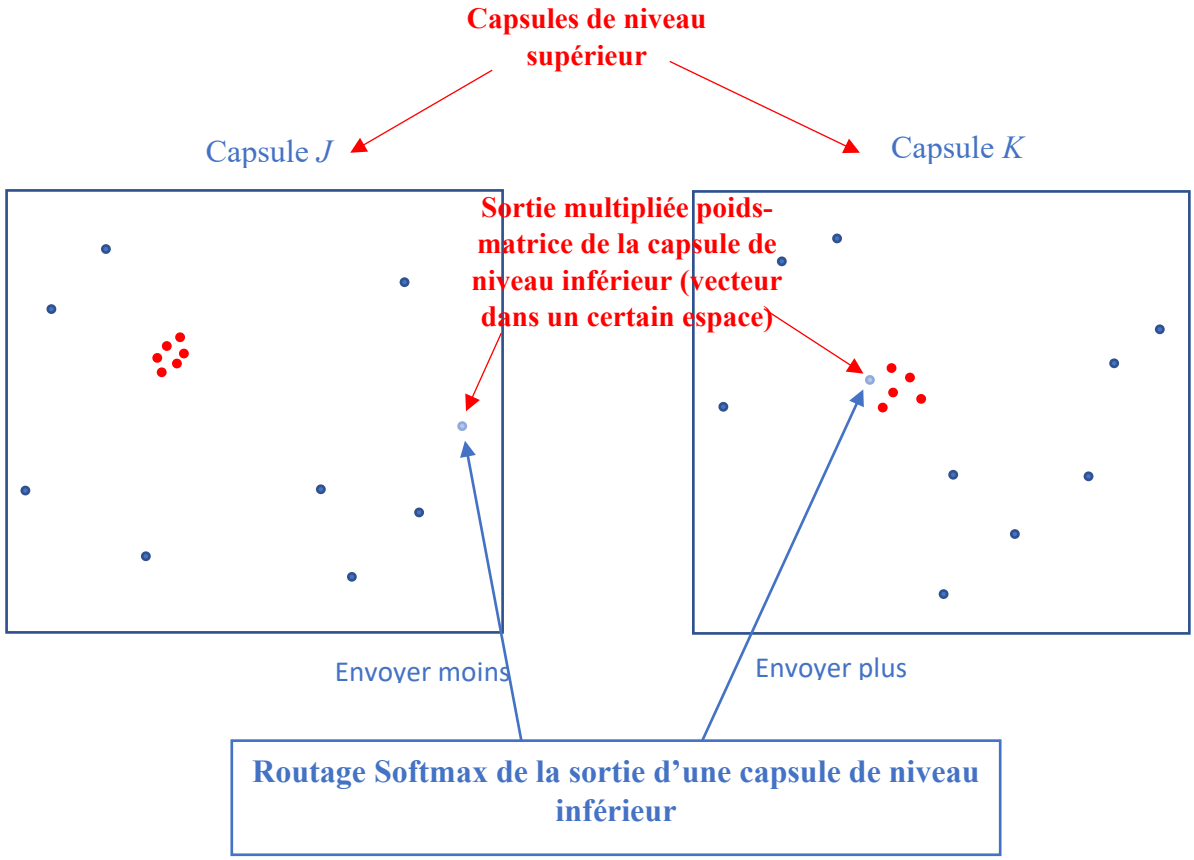


Figure 3. 18. Routage dynamique basé sur l'accord.

La capsule de niveau inférieur enverra son entrée à la capsule de niveau supérieur qui « est d'accord » avec son entrée. C'est l'essence de l'algorithme de Routage dynamique [47].

Dans la figure 3.17, nous avons une capsule de niveau inférieur qui doit « décider » à quelle capsule de niveau supérieur elle enverra sa sortie. Il prendra sa décision en ajustant les poids $\boldsymbol{C}$ qui multiplieront la sortie de cette capsule avant de l'envoyer vers les capsules de niveau supérieur gauche ou droite $\boldsymbol{J}$ et $\boldsymbol{K}$.

Maintenant, les capsules de niveau supérieur ont déjà reçu de nombreux vecteurs d'entrée d'autres capsules de niveau inférieur. Toutes ces entrées sont représentées par des points rouges et bleus. Lorsque ces points se regroupent, cela signifie que les prédictions des capsules de niveau inférieur sont proches les unes des autres. C'est pourquoi, à titre d'exemple, il y a un groupe de points rouges dans les deux capsules $\boldsymbol{J}$ et $\boldsymbol{K}$.

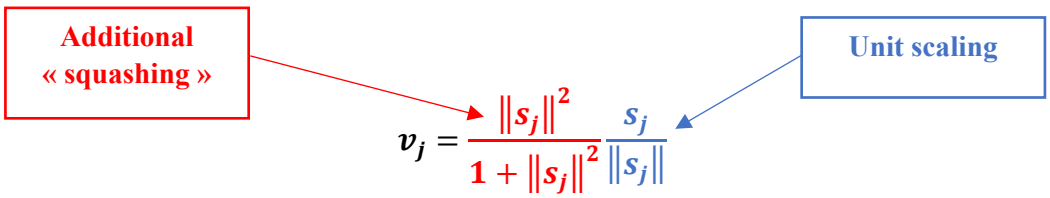
Alors, où notre capsule de niveau inférieur devrait-elle envoyer sa sortie : vers la capsule $\boldsymbol{J}$ ou vers la capsule $\boldsymbol{K}$? La réponse à cette question est l'essence de l'algorithme de routage dynamique. La sortie de la capsule inférieure, lorsqu'elle est multipliée par la matrice correspondante $\boldsymbol{W}$, atterrit loin du groupe rouge des prédictions « correctes » dans la capsule $\boldsymbol{J}$. En revanche, elle atterrira très près des prédictions « vraies » grappe rouge dans la capsule droite $\boldsymbol{K}$. La capsule de niveau inférieur a un mécanisme pour mesurer quelle capsule de niveau supérieur s'adapte le mieux à ses résultats et ajustera automatiquement son poids de telle sorte que le poids $\boldsymbol{C}$ correspondant à la capsule $\boldsymbol{K}$ sera élevé et le poids $\boldsymbol{C}$ correspondant à la capsule $\boldsymbol{J}$ sera faible [47].

c) Somme des vecteurs d'entrée pondérés

Cette étape est similaire au neurone artificiel régulier et représente une combinaison d'entrées. Il n'y ait rien de spécial à propos de cette étape (sauf que c'est une somme de vecteurs et non une somme de scalaires). Nous pouvons donc passer à l'étape suivante [47].

d) « Squash » : nouvelle non-linéarité de vecteur à vecteur

Une autre innovation introduite par CapsNet est la nouvelle fonction d'activation non linéaire qui prend un vecteur, puis « l'écrase » pour qu'il ait une longueur ne dépassant pas 1, mais ne change pas de direction [47].



La non-linéarité écrasée met à l'échelle le vecteur d'entrée sans changer sa direction.

Le côté droit de l'équation (en couleur bleu) met à l'échelle le vecteur d'entrée afin qu'il ait une longueur unitaire et le côté gauche (en couleur rouge) effectue une mise à l'échelle

supplémentaire. N'oubliez pas que la longueur du vecteur de sortie peut être interprétée comme la probabilité qu'une caractéristique donnée soit détectée par la capsule [47].

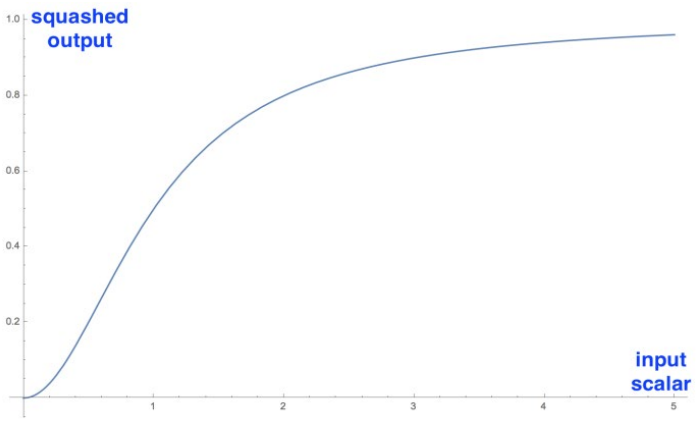


Figure 3. 19. La nouvelle non-linéarité sous sa forme scalaire. En application réelle, la fonction opère sur des vecteurs [47].

Sur la gauche se trouve la fonction d'écrasement appliquée à un vecteur 1D, qui est un scalaire. Nous avons inclus pour démontrer la forme non linéaire intéressante de la fonction.

Cela n'a de sens que de visualiser un cas à une dimension ; dans une application réelle, il prendra un vecteur et produira un vecteur, ce qui serait difficile à visualiser [47].

3.9.2 Routage dynamique entre les capsules

L'algorithme de routage est le suivant [44] :

Algorithme de routage

1 : Procedure ROUTING ($\hat{u}_{j|i}$ **,** r **,** l **)**

2 : for all capsule i in layer l and capsule j in layer $(l+1)$: $b_{ij} \leftarrow 0$

3 : **for** r iterations **do**

4 : for all capsule i in layer l : $c_i \leftarrow softmax(b_i)$

5 : for all capsule j in layer $(l+1)$: $s_j \leftarrow \sum_i c_{ij} \hat{u}_{j|i}$

6 : for all capsule j in layer $(l+1)$: $v_j \leftarrow squash(s_j)$

7 : for all capsule i in layer l and capsule j in layer $(l+1)$:

$$b_{ij} \leftarrow b_{ij} + \hat{u}_{j|i} \cdot v_j$$

return v_j

La première ligne indique que cette procédure prend toutes les capsules à un niveau inférieur l et leurs sorties û, ainsi que le nombre d'itérations de routage r. La toute dernière ligne vous indique que l'algorithme produira la sortie d'une capsule de niveau supérieur v_j. Essentiellement, cet algorithme nous indique comment calculer le passage avant du réseau [48].

Dans la deuxième ligne, il y a un nouveau coefficient b_{ij} . Ce coefficient est simplement une valeur temporaire qui sera mise à jour de manière itérative et, une fois la procédure terminée, sa valeur sera stockée dans c_{ij}. Au début de la formation, la valeur de b_{ij} est initialisé à zéro.

La ligne 3 indique que les étapes 4 à 7 seront répétées r fois (le nombre d'itérations de routage).

L'étape de la ligne 4 calcule la valeur du vecteur c_i qui sont tous les poids d'acheminement pour une capsule de niveau inférieur i. Ceci est fait pour toutes les capsules de niveau inférieur. Pourquoi softmax ? Softmax s'assurera que chaque poids c_{ij} est un nombre non

négatif et leur somme est égale à un. Essentiellement, softmax applique la nature probabiliste des coefficients $\boldsymbol{c_{ij}}$ [48].

À la première itération, la valeur de tous les coefficients $\boldsymbol{c_{ij}}$ sera égal, car sur la ligne deux tous $\boldsymbol{b_{ij}}$ sont mis à zéro. Par exemple, si nous avons 3 capsules de niveau inférieur et 2 capsules de niveau supérieur, alors tout $\boldsymbol{c_{ij}}$ sera égal à 0,5. L'état de tous $\boldsymbol{c_{ij}}$ être égal à l'initialisation de l'algorithme représente l'état de confusion et d'incertitude maximales : les capsules de niveau inférieur n'ont aucune idée des capsules de niveau supérieur qui conviendront le mieux à leur sortie. Bien sûr, à mesure que le processus se répète, ces distributions uniformes changeront [48].

Après tous les poids $\boldsymbol{c_{ij}}$ ont été calculés pour toutes les capsules de niveau inférieur, nous pouvons passer à la ligne 5, où nous examinons les capsules de niveau supérieur. Cette étape calcule une combinaison linéaire de vecteurs d'entrée, pondérée par des coefficients de routage $\boldsymbol{c_{ij}}$, déterminé à l'étape précédente. Intuitivement, cela signifie réduire les vecteurs d'entrée et les ajouter ensemble, ce qui produit un vecteur de sorties $\boldsymbol{s_i}$. Ceci est fait pour toutes les capsules de niveau supérieur [48].

Ensuite, à la ligne 6, les vecteurs de la dernière étape sont passés à travers la non-linéarité de squash (fonction d'activation), qui garantit que la direction du vecteur est préservée, mais que sa longueur ne doit pas dépasser 1. Cette étape produit le vecteur de sortie $\boldsymbol{v_j}$ pour toutes les capsules de niveau supérieur [48].

Pour résumer : les étapes 4 à 6 calculent simplement le rendement des capsules de niveau supérieur. L'étape de la ligne 7 correspond à la mise à jour du poids. Cette étape capture l'essence de l'algorithme de routage. Cette étape examine chaque capsule de niveau supérieur $\boldsymbol{j}$ puis examine chaque entrée et met à jour le poids correspondant $\boldsymbol{b_{ij}}$ selon la formule. La formule indique que la nouvelle valeur de poids est égale à l'ancienne valeur plus le produit scalaire de la sortie actuelle de la capsule $\boldsymbol{j}$ et l'entrée dans cette capsule à partir d'une capsule de niveau inférieur $\boldsymbol{i}$. Le produit scalaire examine la similitude entre l'entrée dans la capsule et la sortie de la capsule. De plus, la capsule de niveau inférieur enverra sa sortie à la capsule de niveau supérieur dont la sortie est similaire. Cette similitude est capturée par le produit scalaire. Après cette étape, l'algorithme recommence à partir de l'étape 3 et répète le processus $\boldsymbol{r}$ fois [48].

Après $\boldsymbol{r}$ fois, toutes les sorties pour les capsules de niveau supérieur ont été calculées et les poids d'acheminement ont été établis. Le passage avant peut continuer au niveau suivant du réseau [48].

Combien d'itérations de routage utiliser ?

Il est recommandé d'utiliser 3 itérations de routage dans la pratique.

3.9.3 L'architecture de CapsNet

Le CapsNet se compose de 2 parties : encodeur et décodeur. Les 3 premières couches sont codeur, et les 3 secondes sont décodeur :

- Couche 1. Couche convolutionnelle.
- Couche 2. Couche PrimaryCaps.
- Couche 3. Couche DataCaps.
- Couche 4. Entièrement connecté # 1.
- Couche 5. Entièrement connecté # 2.
- Couche 6. Entièrement connecté # 3.

3.9.4 Fonction de perte (Perte de marge)

La longueur du vecteur d'instanciation représente la probabilité que l'entité de la capsule est présente dans la scène. Une capsule de niveau supérieur à un long vecteur si et seulement si l'entité associée est présente. Pour permettre plusieurs entités, une distincte perte de marge est calculée pour chaque capsule. Downweighting (Pondération réduite) la perte des entités absentes arrête l'apprentissage de la diminution des longueurs de vecteur d'activité pour toutes les entités. La perte totale est la somme des pertes de toutes les entités. Dans l'exemple de Hinton la fonction de perte est [44] :

$$\boldsymbol{L_k} = \underbrace{\boldsymbol{T_k} \max(0, m^+ - \|\boldsymbol{v_k}\|)^2}_{\text{Classe présente}} + \underbrace{\lambda(1 - \boldsymbol{T_k}) \max(0, \|\boldsymbol{v_k}\| - m^-)^2}_{\text{Classe n'est pas présente}}$$

$$\boldsymbol{T_k} = \begin{Bmatrix} 1, chiffre\ de\ la\ classe\ k\ présent \\ 0, sinon \end{Bmatrix}$$

Ce type de fonction de perte est courant dans les ANNs. Les paramètres m^+ et m^- sont définis de sorte que la longueur ne dépasse pas ou ne se réduit pas, $m^+ = 0{,}9$ et $m^- = 0.1$. La sous-

pondération des poids initiaux pour les classes absentes est contrôlée par λ, avec $\lambda = 0{,}5$ comme choix raisonnable [44].

3.9.5 L'architecture de CAPSNET proposé

Le CapsNet se compose de 2 parties : encodeur et décodeur.

3.9.5.1 Partie 1 : Encodeur

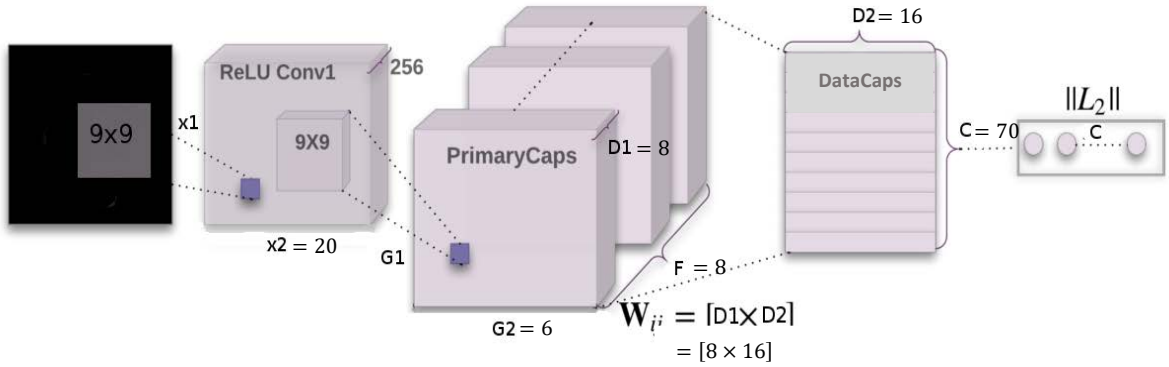


Figure 3. 20. Architecture de partie encodeur de CapsNet que Nous avons appliqué sur la base de données.

La partie encodeur du réseau prend en entrée une image de 28 par 28 chiffres, lettres (majuscules et minuscules) et marques de ponctuation et apprend à l'encoder dans un vecteur à 16 dimensions de paramètres d'instanciation, c'est là que les capsules font leur travail. La sortie du réseau pendant la prédiction est un vecteur à 70 dimensions de longueurs des sorties de DataCaps. Le décodeur a 3 couches : deux d'entre elles sont convolutives et la dernière est entièrement connectée.

a) Couche convolutionnelle

- Entrée : image 28x28 (un canal de couleur).
- Sortie : tenseur 20x20x256.
- Nombre de paramètres : 20992.

Le travail de la couche convolutionnelle est de détecter les caractéristiques de base dans l'image. Dans le CapsNet, la couche convolutionnelle a 256 noyaux avec une taille de 9x9x1 et stride 1, suivis de l'activation ReLU.

Pour calculer le nombre de paramètres, on sait que dans chaque noyau d'une couche convolutive a 1 terme de biais. Par conséquent, cette couche a (9x9 + 1) x256 = 20992 paramètres pouvant être entraînés au total.

b) Couche PrimaryCaps

- Entrée : tenseur 20x20x256.
- Sortie : tenseur 6x6x8x8.
- Nombre de paramètres : 1327168.

Cette couche a 8 capsules primaires dont le travail est de prendre les caractéristiques de base détectées par la couche convolutionnelle et de produire des combinaisons des caractéristiques. La couche a 8 « capsules primaires » qui sont très similaires à la couche convolutionnelle dans leur nature. Chaque capsule applique huit noyaux convolutifs 9x9x256 (avec stride 2) au volume d'entrée 20x20x256 et produit donc un tenseur de sortie 6x6x8. Puisqu'il y a 8 capsules de ce type, le volume de sortie a la forme de 6x6x8x8. En effectuant un calcul similaire à celui de la couche précédente, nous obtenons 1327168 paramètres pouvant être entraînés dans cette couche.

c) Couche DataCaps

- Entrée : tenseur 6x6x8x8.
- Sortie : matrice 16x70.
- Nombre de paramètres : 2580480.

Cette couche a 70 capsules de chiffres, lettres (majuscules et minuscules) et marques de ponctuation, une pour chaque classe. Chaque capsule prend en entrée un tenseur 6x6x8x8. Vous pouvez le considérer comme des vecteurs à 8 dimensions 6x6x8, soit 288 vecteurs d'entrée au total. Selon le fonctionnement interne de la capsule, chacun de ces vecteurs d'entrée obtient sa propre matrice de poids 8x16 qui mappe l'espace d'entrée à 8 dimensions vers l'espace de sortie de la capsule à 16 dimensions. Ainsi, il y a 288 matrices pour chaque capsule, ainsi que 288 coefficients c et 288 b coefficients utilisés dans le routage dynamique. En multipliant : 288 x 8

x 16, nous obtenons 36864 paramètres entraînables par capsule, puis nous multiplions par 70 pour obtenir le nombre final de paramètres pour cette couche.

3.9.5.2 Partie 2 : Décodeur

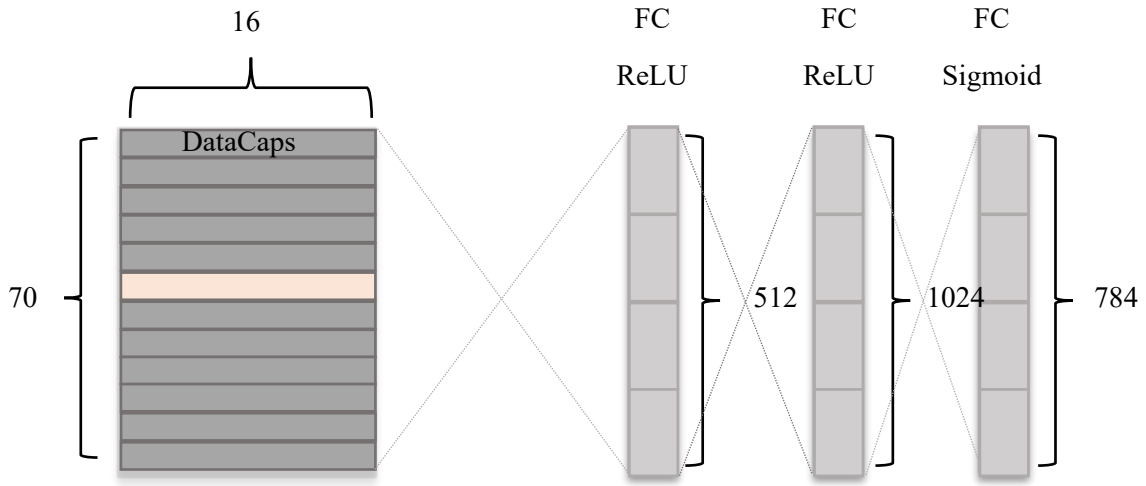


Figure 3. 21. Architecture de partie Décodeur de CapsNet que Nous avons appliqué sur la base de données.

Le décodeur prend un vecteur à 16 dimensions du DataCaps correct et apprend à le décoder en une image d'un chiffre ou lettres (notez qu'il n'utilise que le bon vecteur DataCaps pendant l'entraînement et ignore les mauvais). Le décodeur est utilisé comme régulariseur, il prend la sortie du DataCaps correct comme entrée et apprend à recréer une image de 28 par 28 pixels, la fonction de perte étant la distance euclidienne entre l'image reconstruite et l'image d'entrée. Le décodeur force les capsules à apprendre les fonctionnalités utiles pour reconstruire l'image d'origine. Plus l'image reconstruite est proche de l'image d'entrée.

a) Couche entièrement connecté 1

- Entrée : 16x70.
- Sortie : 512.
- Nombre de paramètres : 537952.

Chaque sortie du niveau inférieur est pondérée et dirigée vers chaque neurone de la couche entièrement connectée en tant qu'entrée. Chaque neurone a également un terme de biais. Pour

cette couche, il y a 16x70 entrées qui sont toutes dirigées vers chacun des 512 neurones de cette couche. Par conséquent, il existe (16x70 + 1) x512 paramètres pouvant être entraînés.

Pour les deux couches suivantes, le calcul est le même : nombre de paramètres = (nombre d'entrées + biais) x nombre de neurones dans la couche. C'est pourquoi il n'y a aucune explication pour les couches 2 et 3 entièrement connectées.

b) Couche entièrement connecté 2

- Entrée : 512.
- Sortie : 1024.
- Nombre de paramètres : 525312.

c) Couche entièrement connecté 3

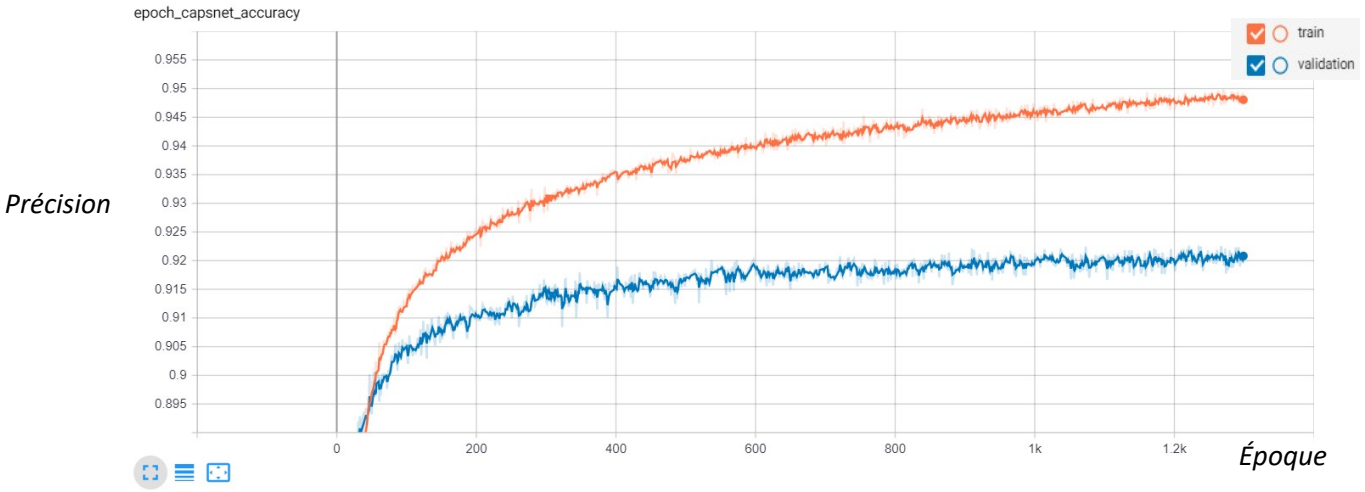
- Entrée : 1024.
- Sortie : 784 (qui après le remodelage rend une image décodée 28x28).
- Nombre de paramètres : 803600.

Nombre total de paramètres dans le réseau : 5831504.

3.10 Résultat et discussion

3.10.1 Résultat obtenu pour le modèle de CAPSNET

Modèle de précision



Modèle de perte

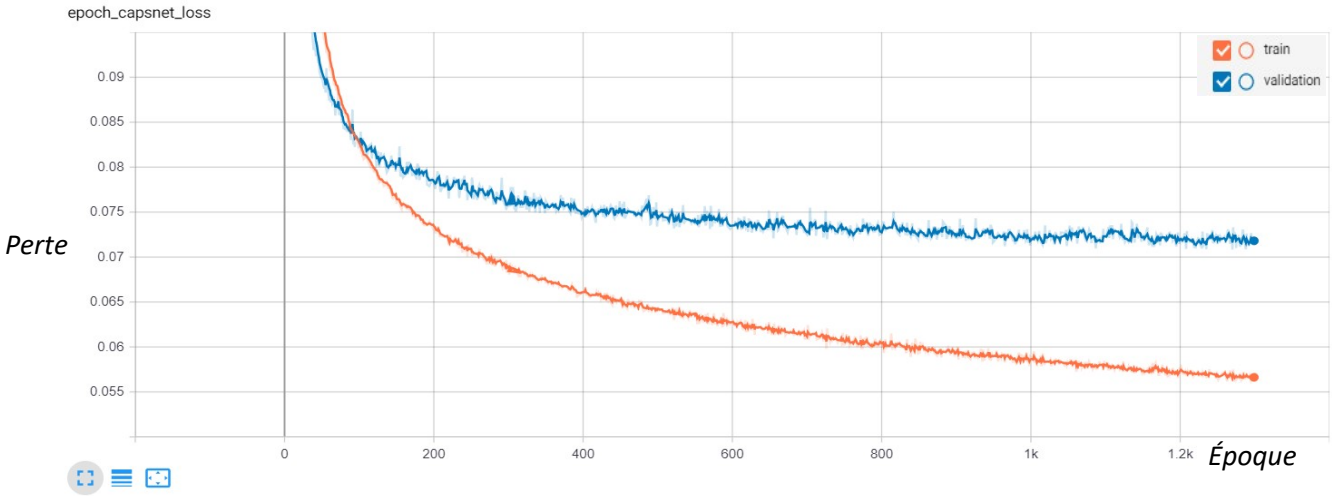


Figure 3. 22. La précision et l'erreur de modèle de CAPSNET.

3.10.2 Tableau de comparaison de résultats

Le tableau ci-dessous montre les différents résultats obtenus sur les deux modèles de deux techniques.

Tableau 3. 6. Les différents résultats obtenus sur les deux modèles de deux techniques.

Modèle Technique de réseau neuronal	Architecture utilisé			Nombre d'époque	Précision obtenue sur la base d'apprentissage	Précision obtenue sur la base de validation	Erreur	Temps d'exécution (En utilisons GPU)
	Couche de convolution	Couche de Pooling	Couche de Fully connected *(FC)*					
Modèle de CNN	06	03	01	2000	91%	88%	20%	(9 heures)
Modèle de CapsNet	03	L'architecture de CapsNet ne contient pas la couche de *Pooling*	04	1300	94%	92%	06%	(10 jours)

Le tableau montre la différence entre les deux modèles. Les résultats obtenus sont exprimés en termes de précision d'apprentissage, de validation, de test et erreur et enfin le temps d'exécution.

Le temps d'exécution est trop couteux. Ceci revient à la grande dimension de la base ce qui nécessite l'utilisation d'un GPU au lieu d'un CPU.

Le taux de précision de modèle de CapsNet est majeur par rapport au taux de précision de modèle de CNN.

Le taux d'erreur de modèle de CNN est trois fois plus que le taux d'erreur de modèle CapsNet.

Finalement, Nous avons constaté que le modèle de CapsNet a eu des résultats meilleurs à celle de modèle de CNN malgré que le nombre d'époque de cette dernière soit plus que le nombre d'époque de modèle de CapsNet.

Par contre l'approche de réseaux de capsules dans l'étape d'apprentissage est beaucoup plus lente que les modèles de réseaux de neurones convolutionnels.

3.11 Conclusion

Nous avons présenté dans ce chapitre Notre deux approches de reconnaissance de caractères basée sur les réseaux de neurones (convolutionnels et capsules), pour cela on a utilisé deux modèles de deux approches et on a montré les différents résultats obtenus en termes de précision et d'erreur. La comparaison des résultats trouvés a montré que le modèle de réseaux de capsules est bien meilleur que le modèle de réseaux de neurones convolutionnels, et que le nombre d'époque, la taille de la base de données sont des facteurs importants pour l'obtention de meilleurs résultats.

Conclusion Générale

La reconnaissance des caractères des sous-titres vidéo joue un rôle crucial dans l'analyse de la vidéo. Dans ce mémoire, Nous avons présentés un système de reconnaissance de caractères de sous-titres vidéo, Se basant sur deux approches robustes d'apprentissage profond qui sont les réseaux de neurones convolutionnel (CNN) et les réseaux de capsules (CapsNet) pour la reconnaissance de caractères.

Premièrement, Notre système capture toutes les séquences de la vidéo à traiter, les séquences vidéo capturé passe par une étape de prétraitement d'images pour éliminer les bruits et sélectionné la région de sous-titrage puis on filtre que les images avec sous-titres vidéo pour passer à l'étape suivante. Ensuite, les images qui contient de sous-titres passe à une étape importante qui est la segmentation, nous avons appliqués une technique de détection de texte en image à base d'un algorithme de projection verticale et horizontale, ce dernier à donner des résultats intéressants. Finalement, on obtient des imagettes de caractères qui vont être traiter par le modèle trainé soit par l'approche de CNN ou bien CapsNet.

Notre système à donner des bons résultats au niveau de reconnaissance de caractères, cela montre l'efficacité de méthodes utilisées. Nous avons effectué aussi une comparaison entre les résultats obtenus de deux approches.

Nous espérons dans l'avenir de contenue notre recherche dans ce domaine, ainsi d'ajouter un graphe d'inférence pour faire la liaison entre les caractères segmentés et d'ajouter aussi des connaissances linguistiques (modèle de langue et dictionnaire) pour prendre en compte le contexte lexical.

Bibliographie

[1] K. E. F. D. a. A. T. Tike Judd, «“Learning to predict where humans look”,» Proceedings of IEEE 12th International Conference on Computer Vision, pp. 2106-2113, 2009.

[2] Bellman, R. 1961, Adaptive Control Processes: A Guided Tour, Princeton University Press, New Jersey.

[3] Blumer, A., A. Ehrenfeucht, D. Haussler et M. Warmuth. 1987, “Occam’srazor”, Inf. Proc. Let., vol. 24, p. 377–380.

[4] Kolmogorov, A. N. 1965, “Three approaches to the quantitative definition of information”, Problems of Information and Transmission, vol. 1, no 1, p. 1–7.

[5] Li, M. et P. Vitànyi. 2008, An Introduction to Kolmogorov Complexity and Its Applications, 3e ´ed., Springer, New York, NY.

[6] Nadaraya, E. A. 1964,“On estimating regression”, Theory of Probability and its Applications, vol. 9, p. 141–142.

[7] Solomonoff, R. J. 1964,“A formaltheoryof inductive inference”, Information and Control, vol. 7, p. 1–22, 224–254. Sutton, R. et A. Barto. 1998, Reinforcement Learning : An Introduction, MIT Press.

[8] Watson, G. S. 1964, “Smooth regression analysis”, Sankhya - The Indian Journal of Statistics, vol. 26, p. 359–372.

[9] Weizenbaum, J. 1966, “ELIZA-a computer program for the study of natural language communication between man and machine”, Commun. ACM, vol. 9, no 1, p. 36–45.

[10] Barber, D. 2011, Bayesian Reasoning and Machine Learning, Cambridge University Press.

[11] Boser, B. E., I. M. Guyon et V. N. Vapnik. 1992, “A training algorithm for optimal margin classifiers”, dans COLT ’92 : Proceedings of the fifth annual workshop on Computational learning theory,ACM,NewYork,NY, USA, p. 144–152.

[12] Breiman, L., J.H. Friedman, R.A. OlshenetC.J. Stone.1984, Classification and Regression Trees, Wadsworth International Group, Belmont, CA.

[13] Cortes, C. etV.Vapnik.1995, “Supportvectornetworks”, Machine Learning, vol. 20, p. 273–297.

[14] Dempster, A. P., N. M. Laird et D. B. Rubin. 1977, “Maximum-likelihood from incomplete data via the EM algorithm”, Journal of Royal Statistical Society B, vol. 39, p. 1–38.

[15] «Hinton, G. E. et T. J. Sejnowski. 1986, “Learning and relearning in Boltzmann machines”, dans Parallel Distributed Processing : Explorations in the Microstructure of Cognition,» . Volume 1 :

Foundations, édité par D. E. Rumelhart et J. L. McClelland, MIT Press, Cambridge, MA, p. 282–317.

[16] Hinton, G. E., T. J. Sejnowski et D. H. Ackley. 1984,“Boltzmann machines : Constraint satisfaction networks thatlearn”, rapport technique TR-CMUCS-84-119, Carnegie-Mellon University, Dept. of Computer Science.

[17] Hornik, K., M. Stinchcombe et H. White. 1989, “Multilayer feedforward networks are universal approximators”, Neural Networks, vol. 2, p. 359– 366.

[18] Parzen, E. 1962, “On the estimation of a probability density function and mode”, Annals of Mathematical Statistics, vol. 33, p. 1064–1076.

[19] Rasmussen, C. E. et C. Williams. 2006, Gaussian Processes for Machine Learning, MIT Press.

[20] Rosenblatt, M. 1956,“Remarks on some nonparametric estimates of a density function”, The Annals of Mathematical Statistics, vol. 27, no 3, p. 832–837.

[21] Rumelhart, D. E., G. E. Hinton et R. J. Williams. 1986, “Learning representations by back-propagating errors”, Nature, vol. 323, p. 533–536.

[22] Scholkopf, B. et A. J. Smola. 2002, Learning with Kernels : Support Vector Machines, Regularization, Optimization and Beyond, MIT Press, Cambridge, MA.

[23] Smolensky, P. 1986, “Information processing in dynamical systems : Foundations of harmony theory”, dans Parallel Distributed Processing, vol. 1, édité par D. E. Rumelhart et J. L. McClelland, chap. 6, MIT Press, Cambridge, p. 194–281.

[24] Vapnik, V. 1998, Statistical Learning Theory, Wiley, Lecture Notes in Economics and Mathematical Systems, volume 454.

[25] Wainwright, M. J. et M. I. Jordan. 2008, “Graphical models, exponential families, and variational inference”, Foundations and Trends in Machine Learning, vol. 1, no 1-2, p. 1–305.

[26] ‘’Apprentissage machine efficace : théorie et pratique’’, par Olivier Delalleau , Département d’informatique et de recherche opérationnelle Faculté des arts et des sciences.

[27] Watanabe, S. Watanabe, “Pattern recognition: Human and Mechanical”, New York Wiley, 1985.

[28] Theodoridis et Koutroumbas, S. Theodoridis, K. Koutroumbas, “Pattern Recognition, Second Edition”, Academic Press, Elsevier, 2003.

[29] [En ligne]. Available: https://interstices.info/jcms/p_88807/marvin-minsky-un-des-cerveaux-de-l-intelligenceartificielle .

[30] V. Bjorn, “One Finger at a Time: Best Practices for Biometric Security,” Banking Information Source (Document ID: 1697301411), April, 2009.

[31] Chung et al, K.W. Cheung, J.T. Kwok, M.H. Law, K.C. Tsui,"Mining customer product ratings for personalized marketing", Decision Support Systems, vol. 35, issue 2, pp. 231-243, 2003.

[32] Kaastra et Boyd, I. Kaastra, M. Boyd, "Designing a neural network for forecasting financial and economic time series", Neurocomputing. Vol. 10, no. 3, pp. 215-236. 1996.

[33] Gupta et al, S.K. Gupta, W.C. Regli, D.S. Nau, "Manufacturing Feature Instances: Which Ones to Recognize?", Symposium on Solid Modeling and Applications, pp. 141152, 1995.

[34] Hippert et al, H.S. Hippert, C.E. Pedreira, R.C. Souza, "Neural networks for shortterm load forecasting: a review and evaluation", IEEE Transactions on Power Systems, vol. 16, Part 1, pp. 44-55, 2001.

[35] Munich et Perona, M.E. Munich, P. Perona, "Visual Input for Pen-Based Computers", IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 24, issue 3, 2002.

[36] Yang et al, M.H. Yang, D.J. Kriegman, N. Ahuja, "Detecting Faces in Images: A 51 Survey", IEEE Transactions On Pattern Analysis And Machine Intelligence PAMI, vol.24; part 1, pp. 34-58, 2002.

[37] Theodoridis et Koutroumbas, S. Theodoridis, K. Koutroumbas, “Pattern Recognition, Second Edition”, Academic Press, Elsevier, 2003.

[38] [En ligne]. Available: https://www.interstices.info/jcms/c_5952/histoire-du-traitement-d-images.

[39] ‘’Mise au Point d’une Application de Reconnaissance de Formes’’, Mémoire de fin d’études pour l’obtention du diplôme de Master en Informatique, DJABEUR DJEZZAR Mohammed Rafik, BENKADA Feth-a.

[40] Kullback, S. 1959, Information Theory and Statistics, Wiley, New York.

[41] Satish S. Hiremath and K.V. Suresh. CHARACTER RECOGNITION OF VIDEO SUBTITLES. Department of Electronic and Communication Engineering, Siddaganga Institute of Technology, India,November 2016.

[42] C. Dorai, H. Aradhye, and J.-C. Shim. End-to-end video text recognition for multimedia content analysis. In ICME, pages 601–604, 2001.

[43] Khaoula Elagouni, Christophe Garcia, Pascale Sébillot. Reconnaissance automatique de texte dans des vidéos à l’aide d’un OCR et de connaissances linguistiques. GRETSI, Sep 2011, Bordeaux, France.

[44] S. Sabour, N. Frosst et G. E. Hinton, «Dynamic Routing Between Capsules,» (2017-10-26).

[45] Q. C. ,. C. W. ,. Z. L. Wangsheng Zhu, « A segmentation algorithm based on imageprojection for complex text layout,» October 2017 .

[46] A. A. e. S. Amidi, «CS 230 – Apprentissage profond Petites astuces,» Université de stanford, 6 janvier 2019.

[47] M. Pechyonkin, «Understanding Hinton’s Capsule Networks. Part 2. How Capsules Work.,» 15 Nov 2017. [En ligne]. Available: https://pechyonkin.me/capsules-2/.

[48] M. Pechyonkin, «Understanding Hinton's Capsule Networks. Part 3. Dynamic Routing Between Capsules,» 29 Nov 2017. [En ligne]. Available: https://pechyonkin.me/capsules-3/.

[49] A. A. e. S. Amidi, «CS 230 – Apprentissage profond : Réseaux de neurones Convolutionnels,» Université de stanford, 6 janvier 2019.

[50] « Classification des images avec les réseaux de neurones Convolutionnels », Mémoire de fin d'études pour l'obtention du diplôme de Master en informatique, M. Mokri Mohammed Zakaria.

[51] [En ligne]. Available: https://pypi.org/project/Pillow/.

[52] [En ligne]. Available: https://pypi.org/project/tensorflow-gpu/.

[53] [En ligne]. Available: https://fr.wikipedia.org/wiki/Keras.

[54] [En ligne]. Available: https://fr.wikipedia.org/wiki/Tkinter.

[55] [En ligne]. Available: https://www.jetbrains.com/fr-fr/pycharm/.

[56] [En ligne]. Available: https://fr.wikipedia.org/wiki/OpenCV.

[57] [En ligne]. Available: https://docs.python.org/fr/3/tutorial/.

[58] ''Using Convolutional Neural Networks for Image Recognition'', Samer Hijazi, Rishi Kumar,Cadence.