UNIVERSITÉ DU QUÉBEC À MONTRÉAL

EXTRACTION ET COMPRÉHENSION DE TEXTE DANS LES FILMS

MÉMOIRE
PRÉSENTÉ
COMME EXIGENCE PARTIELLE
DE LA MAÎTRISE EN INFORMATIQUE

PAR
SÉBASTIEN TESTEAU

MARS 2022

TABLE DES MATIÈRES

LISTE DES FIGURES

LISTE DES TABLEAUX

RÉSUMÉ

Ce mémoire s'intéresse à la compréhension de texte dans les films et plus généralement à la détection et à la reconnaissance de texte dans les scènes naturelles. Il explore les modèles de langage pour la correction du texte reconnu et s'intéresse aussi à l'indexation et la recherche de contenus dans ce texte. L'idée est de pouvoir reconnaître le texte dans de grandes collections de films et de pouvoir ainsi faire des recherches de contenus particuliers.

Nous proposons pour cela une méthode performante d'extraction d'images dans les fichiers vidéo en suggérant un modèle pour la détection de texte, un modèle de reconnaissance de texte, une suite d'outils pour colliger et corriger le texte reconnu et finalement l'indexation et la recherche de ce texte dans un moteur de recherche.

Outre l'innovation d'unifier les modèles pour la compréhension de texte dans les films, ce mémoire contribue aussi à améliorer l'accessibilité des oeuvres de l'Office national du film du Canada (ONF) et la découvrabilité du patrimoine cinématographique Canadien.

Ce mémoire se base sur la création et l'annotation d'un jeu de données *ONF-Text* pour identifier les modèles les plus performants pour l'ONF pour la détection, la reconnaissance et la recherche de texte dans les films.

Mots-clés : Détection de texte, reconnaissance de texte, segmentation de texte, apprentissage profond, apprentissage automatique, traitement automatique du langage naturel, vision par ordinateur

INTRODUCTION

Depuis l'avènement de la révolution numérique, l'industrie du cinéma déploie d'énormes efforts pour s'y adapter et en tirer profit. Ses balbutiements s'effectuent d'abord dans la postproduction et plus particulièrement dans les effets spéciaux. Ainsi, dès 1982, le film *Tron* mélange des images filmées et numériques. Au début des années 2000, on assiste à l'avènement du numérique dans les salles de cinéma avec la sortie du film *Star Wars - La menace fantôme*. Depuis 2016, plus de 98% des écrans du parc mondial sont équipés en cinéma numérique (Swartz, 2005).

Ce nouveau médium complique passablement la conservation d'un patrimoine déjà fort mal en point. Selon *The Film Foundation* , un institut créé par Martin Scorsese pour la préservation et la sauvegarde de films importants, une proportion de 50% des films produits avant 1950 et 90% de ceux produits avant 1929, aux États-Unis seulement, sont irrécupérables. L'avènement du numérique a tout d'abord ajouté un peu de confusion. Que faire de tous ces films, anciens et nouveaux, sous des formats différents ? Comment préserver les films et assurer leur pérennité ?

L'Office national du film du Canada n'a pas échappé à ce défi. Depuis 2010, la majorité des productions sont entièrement numériques et dans le cadre de son plan de numérisation, entrepris en 2008, il a déjà numérisé une grande partie des actifs de sa collection dans le but de les préserver, de les exploiter et de les rendre accessibles à tous. À ce jour, plus de 8000 films sur un total d'environ 14 000 œuvres ont été numérisés (Fournier, 2019).

Le plan de numérisation et de conservation de l'Office national du film du Canada est un puissant moteur d'innovation et sa réalisation a mis de l'avant plusieurs défis auxquels l'organisation fait face.

L'un d'entre eux est la numérisation de films ou de vidéos ayant des sous-titres incrustés (voir figure 0.3), car ces derniers constituent un énorme potentiel de recherche. Malheureusement, il n'existe pas de façon simple de les récupérer dans l'image sans faire de la détection et de la reconnaissance de texte dans les films. L'intelligence artificielle poussée par l'efficacité des nouveaux processeurs graphiques et de l'apprentissage profond ouvre de nouvelles possibilités.

Jusqu'ici cependant, la détection et la reconnaissance de texte dans les images restent un problème irrésolu en informatique. C'est une problématique plus complexe que la reconnaissance de texte (OCR) dans les documents puisque le texte est sujet à énormément de variations telles que le changement de police, de taille, d'opacité, d'arrière-plan, d'orientation et d'éclairage (Huang *et al.*, 2014).

HISTOIRE

La première machine à OCR fut créée par Gustav Tauschek en 1929 (Shruthi et Verma, 2021). Toutefois, c'est en 1950 qu'on voit apparaître la première application commerciale de reconnaissance de caractères lorsque David Shephard crée « Gismo » dans son grenier (voir figure 0.1). Plus tard, pour faciliter son application, il crée la police «Farrington B7» sur une serviette de table au Waldorf-Astoria (Shephard, 2021). Cette police est d'ailleurs encore utilisée aujourd'hui sur les cartes de crédit.

En 1965, le service postal des États-Unis, le USPS, commence à utiliser le *Multiline Optical-character Reader* pour trier automatiquement le courrier.

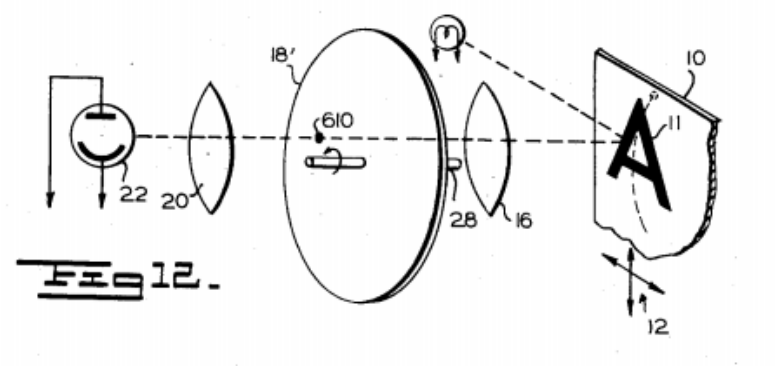

Figure 0.1: Schéma de la machine Gismo OCR extrait de Shephard (2021)

En 2005, Hewlett Packard et l'Université du Nevada libèrent les droits sur leur logiciel ORC Tesseract qui devient la solution gratuite la plus populaire. Elle est aujourd'hui développée et distribuée par Google.

La détection de texte est aujourd'hui un domaine très populaire et énormément de recherches en vision par ordinateur et en apprentissage machine essaient d'y trouver de nouvelles solutions applicables à ce marché en expansion (Lienhart, 1997; Coates *et al.*, 2011; Neumann et Matas, 2012; Gómez et Karatzas, 2015; Shi *et al.*, 2015; Gupta *et al.*, 2016; Yin *et al.*, 2016; Borisyuk *et al.*, 2018; Liao *et al.*, 2019; Ye *et al.*, 2020; Wang *et al.*, 2019). Par exemple, avec l'avènement des autos autonomes, on doit pouvoir décoder les panneaux routiers, les policiers doivent pouvoir reconnaître les plaques d'immatriculation pour donner une amende ou identifier les automobiles volées, ou encore, comme dans notre recherche pouvoir chercher du texte dans des banques de films.

Dans ce contexte, l'ICDAR (International Conference on Document Analysis and Recognition) organise depuis 2003 une compétition de «Robust reading». Elle en est maintenant à sa 5e édition. La "lecture robuste" s'intéresse à la détection et à la reconnaissance de texte dans les images, mais au sens large. Elle fait référence à des techniques et à des méthodologies qui ont été développées pour des conteneurs

de texte autres que des documents papier numérisés tel que des images et des vidéos. En 2013, il y a eu 2633 participants, et en 2015, plus de 5292 (ICD, 2015). La meilleur solution, obtenait à cet époque 49.84% de F-Mesure dans la détection de texte.

DÉFIS

Plusieurs raisons expliquent que la détection et la reconnaissance de texte sont difficiles à réaliser. Par exemple, outre les documents classiques, il existe plusieurs types d'images : entièrement numérique (mémé, affiche, etc.), des scènes naturelles (photos). Il existe aussi des vidéos qui peuvent eux aussi être entièrement numériques, des animations ou encore des scènes naturelles.

Images

Dans les images de scènes naturelles, comme l'illustre la figure 0.2, il n'est pas rare de retrouver des panneaux, des logos, des affiches, des adresses, etc. Leur détection et leur reconnaissance sont particulièrement complexes dans cet environnement.

Films et Vidéos

La reconnaissance de texte dans les films et les vidéos fait face à un autre problème. Dans les années 1980, âge d'or des bandes magnétiques, les formats différents se sont multipliés avant que le numérique n'y mette fin. La multiplicité de ces formats et des technologies qui les supportent ne simplifie pas notre travail.

Généralement, une grande partie du texte dans les films se retrouve dans le sous-titrage, le générique et le sous-titrage pour malentendant. Mais on trouve aussi des informations textuelles intéressantes dans les scènes naturelles. Nous pouvons

Figure 0.2: Exemple de texte dans les scènes naturelles extrait de Liao *et al.* (2018)

penser à des panneaux de circulation, des banderoles, etc.

À l'époque de la pellicule, on gravait les sous-titres dans le film argentique avec un procédé chimique (Wikimedia Foundation, 2021). Les sous-titres perçaient une couche de paraffine où l'émulsion était brûlée par un acide. Ce procédé faisait en sorte que le texte avait souvent un mauvais contraste ainsi qu'une faible opacité comme l'illustre la figure 0.3.

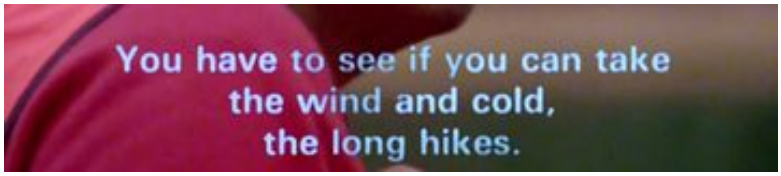

Figure 0.3: Exemple de sous-titre gravé dans un film extrait de La bête lumineuse par Pierre Perrault (ONF)

À l'époque des vidéos, les sous-titres étaient brûlés dans les images avec un contraste clair comme l'illustre la figure 0.4. Dans les années 1980, on utilisait

le montage linéaire pour incruster les sous-titres sur une bande magnétique. Aujourd'hui, on peut incruster des sous-titres à l'aide de logiciel de montage. De nos jours, les films entièrement numériques ne brûlent pas le texte dans l'image, ils ajoutent une piste texte au film ou encore un fichier de sous-titres à part.

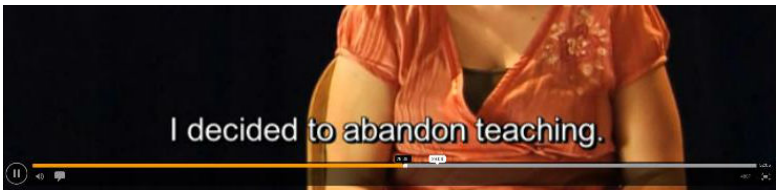

Figure 0.4: Exemple de sous-titre incrusté dans la vidéo pris dans les archives de l'ONF

Taille des données

La taille des données est également importante. Dans notre cas, il faut considérer l'importance de la collection de l'ONF. Si nous prenons pour hypothèse que nous avons 10 000 films à traiter d'une longueur moyenne de 30 minutes et que nous devons extraire 10 images à chaque seconde, nous obtenons 90 000 000 images. Si notre solution prend 30 secondes par images, il nous faudra plusieurs années pour traiter la collection.

Problématique

Pour toutes ces raisons, détecter et reconnaître le texte dans les images est un problème difficile à résoudre. Mais il y a plus : les sous-titres changent de position et de longueur entre les scènes. Par exemple, pour ne pas cacher une information importante, le sous-titre peut apparaître au haut de l'image. À l'ONF, il existe plusieurs films bilingues et, pour certaines versions, le texte peut changer de langue. Parfois, le texte peut avoir plusieurs polices, plusieurs tailles, plusieurs

espacements, plusieurs couleurs, différents niveaux d'opacités et varier en luminosité. En plus, l'arrière-plan change continuellement dans un film et peut avoir la même couleur que le texte comme l'illustre la figure 0.5.

Figure 0.5: Exemple d'un problème de contraste du texte pris dans les archives de l'ONF

Question de recherche

Notre objectif de recherche pour faciliter l'accès et la découvrabilité de la collection de l'Office national du film du Canada (ONF) était confronté à toutes les difficultés que nous venons d'évoquer. Mais nous savions également que l'utilisation des technologies à base d'intelligence artificielle (IA) pouvait nous apporter une aide considérable. Depuis plusieurs années de nombreux essais avaient été effectués pour solutionner ces problématiques et nous comptions bien nous en servir pour trouver nos propres solutions. D'une part, il fallait nous concentrer sur la découverte d'information textuelle et d'autre part, fournir des outils de recherche capables de sonder la collection de l'ONF.

Dans un premier temps, nous avons fabriqué un jeu de données de films ayant du texte pour la conception et l'entraînement d'un modèle, créé une plate-forme de traitement de bout en bout pour la détection et la reconnaissance dans les films. Nous avons comparé les solutions les plus récentes en détection de texte, en reconnaissance de texte et en post traitement de la reconnaissance. Nous avons créé un algorithme qui regroupe le texte par similitude et par proximité temporelle. Finalement, nous avons créé un outil de recherche et de visualisation pour en exploiter les résultats.

Plan du mémoire

Le plan du mémoire que nous présentons maintenant nous permettra de visualiser toutes les étapes de notre cheminement. Dans le chapitre 1, nous décrivons brièvement les concepts de l'apprentissage automatique et de l'apprentissage profond utilisés dans nos travaux. Ensuite, nous établissons dans le chapitre 2 une revue de l'état de l'art exhaustif de la reconnaissance d'objet, de la détection et de la localisation de texte dans les images, de la reconnaissance de texte et du traitement automatique du langage naturel. Ceci nous permet de donner un aperçu des technologies pouvant contribuer à notre objectif de recherche.

Au chapitre 3, nous expliquons la méthodologie utilisée dans le domaine de l'IA et plus particulièrement de la détection et de la reconnaissance de texte dans les images. Nous expliquons aussi la méthodologie que nous avons suivie dans ce mémoire. Nous faisons ensuite un inventaire des données auxquelles nous avons accès, des données utilisées dans la littérature et des données que nous avons annotées ou encore générées automatiquement. Nous concluons par les problèmes que nous avons rencontrés en cours de route.
Le chapitre 4 contient les expériences réalisées ainsi que leur analyse.

Compte tenu de toutes ces étapes, les contributions de ce mémoire sont multiples : nous créons un jeu de données pour la détection et la reconnaissance des textes dans les films, nous créons une plate-forme de traitement où l'on fait l'ingestion, l'extraction d'images, la détection de texte, la reconnaissance de texte, le post-traitement de la reconnaissance de texte et finalement l'indexation pour utilisation dans un moteur de recherche. Pour conclure, nous mettons à la disposition des usagers un système de recherche avec lequel il est possible de rechercher et visualiser les résultats. Enfin, la conclusion fait une synthèse des travaux réalisés dans ce mémoire et propose de nouvelles pistes d'expérimentations pouvant permettre des améliorations.

CHAPITRE I

INTRODUCTION À L'APPRENTISSAGE AUTOMATIQUE

1.1 Apprentissage automatique

1.1.1 Définition

L'apprentissage automatique est une branche de l'intelligence artificielle qui repose sur un ensemble d'algorithmes qui permettent aux ordinateurs d'apprendre à partir de données. Ces algorithmes peuvent ainsi, sans être explicitement programmés pour cela, prendre des décisions ou faire des prédictions.

Il y a donc deux phases dans les algorithmes d'apprentissage automatique, l'entraînement pour construire un modèle et le déploiement du modèle qui permet de résoudre les tâches associées à de nouvelles données. L'apprentissage automatique est généralement scindé en différents types décrits ci-après.

1.1.2 Apprentissage non supervisé

Lorsque les données ne sont pas annotées, il est possible de découvrir la structure des données en utilisant des algorithmes d'apprentissage non supervisé. Un des buts de l'apprentissage non supervisé est justement de créer une représentation des données d'entrée qui pourra par la suite être utile dans un scénario

d'apprentissage supervisé ou encore simplement d'explorer les données. Il existe plusieurs problèmes d'apprentissage non supervisé, nous allons en présenter deux : le partitionnement et l'extraction de caractéristiques.

Méthode par partitionnement

Les algorithmes de partitionnement permettent de regrouper des objets par similitude et donc d'identifier des groupes différents dans les données. Par exemple, un algorithme de partitionnement pourrait créer des groupes selon les préférences de visionnement d'usager d'un site de vidéos en ligne en se basant sur l'historique d'écoute.

L'algorithme le plus connu de partitionnement est probablement celui des K moyennes (*K-means clustering*) (Lloyd, 1982). Il s'agit d'un algorithme itératif qui débute en initialisant aléatoirement la position des centroïdes dans un plan. L'algorithme associe chaque donnée à son centroïde le plus proche pour créer des regroupements et la moyenne des descripteurs d'un groupe définit la position du centroïde.

Figure 1.1: Exemple de groupage obtenu grâce à l'algorithme des K moyennes extrait de Larochelle (2009)

Ainsi, K moyennes alterne itérativement entre 1) regrouper chaque objet autour du centroïde le plus proche et 2) replacer chaque centroïde selon la moyenne des descripteurs de son groupe, comme l'illustre la figure 1.1.

Extraction de caractéristiques

L'objectif de l'extraction de caractéristiques est d'apprendre une représentation plus utile que la représentation originelle. En général, on l'utilise pour améliorer un autre algorithme d'apprentissage.

Il existe plusieurs façons d'extraire ces caractéristiques (Larochelle, 2009) et une technique simple est de choisir les éléments les plus utiles. Par exemple, si nous avons un fichier de données sur le cancer du poumon, la colonne fumeurs sera beaucoup plus utilisée que le sexe du patient. On appelle cette approche la sélection de caractéristiques.

1.1.3 Apprentissage supervisé

Si les données sont annotées, c'est-à-dire qu'il existe un y pour un élément x tel que $x \in X$ et $y \in Y$, alors, les algorithmes d'apprentissage supervisé peuvent apprendre une fonction paramétrée $F : X \to Y$ qui prédit un résultat y selon une entrée x. En d'autres mots, de manière analogue à des enfants qui apprennent le nom d'animaux ($y \in Y$) à partir d'images ($x \in X$) en les mémorisant, l'algorithme est entraîné par un jeu de données étiqueté. Ainsi, un algorithme de classification multiclasse sera en mesure de reconnaître si une image est de classe chien ou chat par exemple (Larochelle, 2009).

1.1.4 Apprentissage semi-supervisé

Lorsque les données ne sont qu'en partie annotées, alors, les algorithmes d'apprentissage semi-supervisé peuvent utiliser les données non annotées pour compléter l'apprentissage supervisé. Par exemple, dans l'**autoapprentissage**, le classifieur est entraîné avec la partie annotée des données, et ensuite, pour les données non

étiquetées, l'algorithme peut utiliser un modèle pour les annoter automatiquement, en prenant soin de n'utiliser que les résultats ayant obtenu un haut taux de succès (Goodfellow *et al.*, 2016).

1.1.5 Apprentissage par renforcement

Lorsque les données sont dynamiques, les algorithmes d'apprentissage par renforcement (AR) tentent de trouver les solutions optimales, pour un agent autonome, à partir d'expériences. L'algorithme AR tente d'optimiser les récompenses selon les décisions que prend l'agent (Goodfellow *et al.*, 2016). Par exemple, lors de la compétition Donkey (`https://www.donkeycar.com`), une petite auto téléguidée par ordinateur utilise une caméra pour naviguer sur une piste de course. L'algorithme AR tente alors de maximiser la vélocité de l'auto, en donnant des récompenses si elle reste dans les limites de la piste de course et lorsqu'elle sort, elle est pénalisée. Ainsi, l'AR apprend par essais-erreurs.

1.1.6 Apprentissage par transfert

Le transfert est la capacité d'un système à reconnaître et appliquer des connaissances déjà apprises sur des données ou des tâches différentes. Il est très utile de pouvoir utiliser les capacités d'un modèle éprouvé sans avoir a le ré entraîner et de pouvoir bâtir à partir de là. Par exemple, la librairie Pytorch (Wallach *et al.*, 2019) permet d'utiliser des modèles de réseaux de neurones convolutifs pré entraînés sur des objets et l'on peut les recycler sur des tâches de détection de caractères (Larochelle, 2009).

1.1.7 Minimisation du risque empirique (MRE)

Dans le contexte de l'apprentissage supervisé, un algorithme crée un modèle en examinant de nombreux exemples, puis tente de trouver un modèle qui minimise une certaine fonction de perte. Cette opération s'appelle la minimisation du risque empirique.

Lorsque l'algorithme effectue une mauvaise prédiction, il retourne un nombre qui indique le degré de justesse. Si la prédiction est parfaite, la perte est nulle, sinon la perte est supérieure à zéro. Le but d'entraîner un modèle est donc de trouver un ensemble de pondération, qui en moyenne sur tous les exemples, minimise le plus possible la perte.

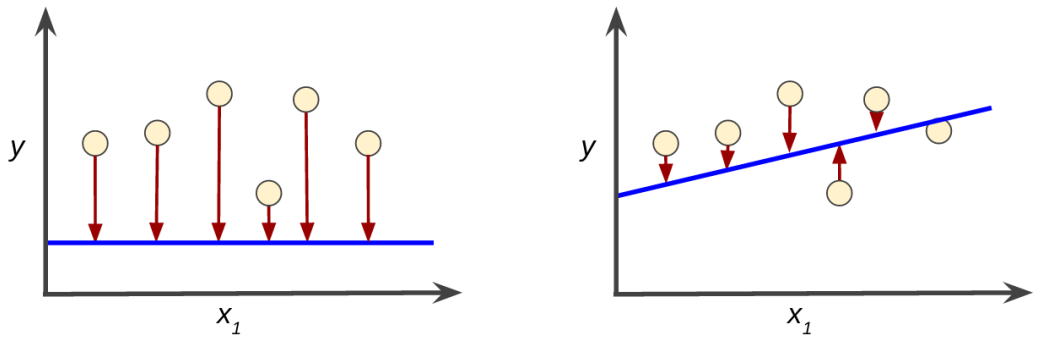

Figure 1.2: Exemple de régression linéaire. La ligne bleue représente le modèle et les données sont le cercle. Extrait de Google (2021)

Dans la figure 1.2, il est facile de se représenter la perte dans un modèle de prédiction. Ainsi, à gauche, on remarque des pertes élevées par rapport à la ligne bleue tandis qu'à droite le modèle est plus performant.

Le risque empirique L_{MRE} d'une fonction f sur l'échantillon $S = \{(x_1, y_1), ..., (x_l, y_l)\}$ est la moyenne de la fonction de perte calculée sur S :

$$R_{MRE}(f) = \frac{1}{n}\sum_{i=1}^{n} L(f(x_i), y_i) \tag{1.1}$$

Où $R_{MRE}(f)$ correspond au risque empirique et $L(f(x_i), y_i)$ représente la fonction de perte sur y_i la prédiction et x_i la donnée réelle.

Il est important de noter que le risque n'est pas l'erreur. En effet, le risque est le coût associé à la procédure d'optimisation et l'erreur est la valeur que l'on souhaite véritablement minimiser. Ainsi, l'erreur n'est pas toujours facile à optimiser, comme dans les tâches de classification. Pour pallier ce problème, on utilise donc des fonctions de coûts telles que le log-vraisemblance qui est optimisable.

Il existe toutefois plusieurs raisons pour lesquelles l'apprentissage ne fonctionnera pas, parmi lesquelles il y a la nature non déterministe d'un problème ou encore la difficulté à minimiser le MRE ou le mauvais échantillonnage des données.

D'ailleurs, dans un réseau de neurones artificiels, la procédure de minimisation se fait généralement avec l'algorithme de la descente de gradient (SGD pour *Stochastic Gradient Descent*) que nous allons explorer dans la section suivante.

1.1.8 La généralisation

Le but de l'apprentissage est de fournir un modèle qui sera capable de solutionner un problème. Pour ce faire, le modèle doit résoudre le problème pas seulement sur les données entraînées, mais aussi sur de nouvelles. On doit ainsi minimiser l'erreur sur les nouvelles données. On appelle cela la généralisation, soit la performance du modèle sur de nouvelles données. Ainsi, comme nous l'avons dans la section précédente 1.1.7, nous utilisons dans la classification le risque empirique pour mesurer cette performance (Larochelle, 2009).

Normalement, lorsqu'on entraîne un modèle, on utilise un sous-ensemble de données pour l'entraînement et un autre pour les tests. Ceci permet d'obtenir une mesure indépendante de l'erreur empirique. D'ailleurs au début de l'entraînement l'erreur sur les deux ensemble baisse, on parle alors de sous-apprentissage. Inversement, lorsque le modèle à trop appris sur les données d'entraînement, on note que l'erreur sur les données de test augmente, on parle alors de surapprentissage, comme on peut le remarque à la figure 1.3

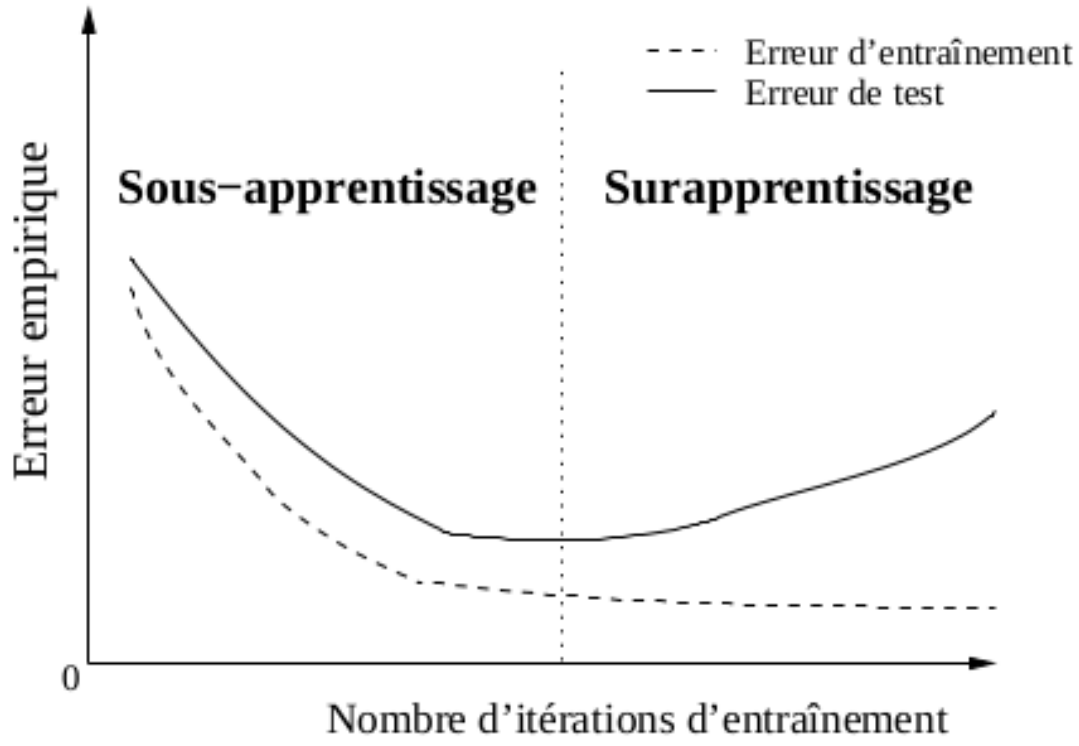

Figure 1.3: Exemple de surappentissage et de sous-apprentissage extrait de Larochelle (2009)

Nous pouvons contrôler si un modèle est plus susceptible de sous-apprendre ou surapprendre en modifiant sa capacité. La capacité d'un modèle, c'est sa capacité à s'adapter à une grande variété de fonctions. Les modèles à faible capacité auront de la difficulté à apprendre les données d'entraînement et les modèles avec de

grandes capacités pourront facilement surapprendre les données en se spécialisant trop sur certaines propriétés de l'ensemble d'entraînement.

1.1.9 La régularisation

Il existe plusieurs solutions au problème de surapprentissage, la régularisation en est une et consiste à ajouter à la fonction de coût un terme de régularisation qui viendra contrôler les valeurs des poids pendant l'apprentissage. Plus les poids ont de grandes valeurs, plus la régularisation va les pénaliser (Goodfellow *et al.*, 2016).

1.2 Réseaux de neurones artificiels

Nous allons dans cette partie nous intéresser aux réseaux de neurones et à leurs nombreux variants. Tout d'abord c'est Rosenblatt (1958) qui créa le premier perceptron à une couche qui permet de classifier linéairement des données. Les réseaux neuronaux sont une famille d'algorithmes qui permettent l'approximation de fonctions complexes.

Les réseaux de neurones constituent aujourd'hui une des techniques de classification les plus utilisées et performantes, surtout en ce qui concerne les problèmes dans le champ d'études de ce mémoire, soit la détection, la localisation et la reconnaissance de texte.

Ainsi, un perceptron à n entrées $(x_1, \ldots, x_n)$ et à une seule sortie o est défini par la donnée de n poids $(w_1, \ldots, w_n)$ et un biais (ou seuil) θ par :

$$o = \begin{cases} 1 & \text{si} \quad \sum_{i=1}^{n} w_i x_i > \theta \\ 0 & \text{sinon} \end{cases} \tag{1.2}$$

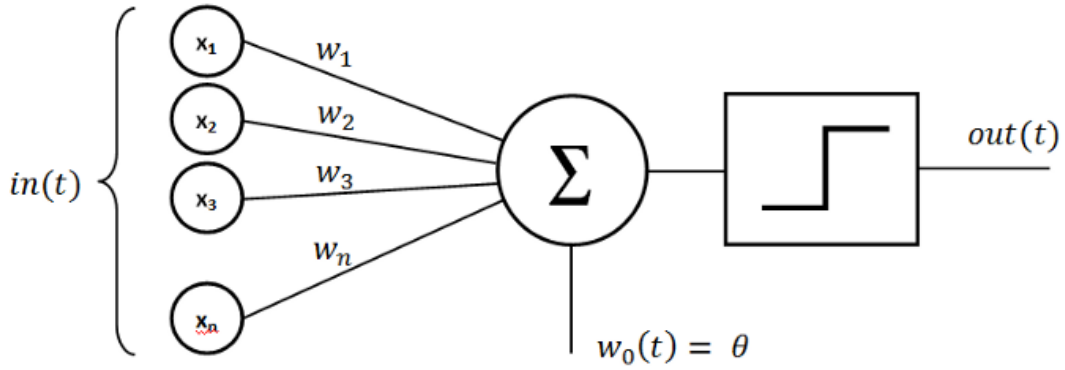

Figure 1.4: Exemple d'un perceptron extrait de Wikimedia Foundation (2013)

Le perceptron est incapable de résoudre de manière satisfaisante un problème de classification si la frontière entre les régions de l'espace d'entrée associées aux différentes classes est non linéaire comme on peut l'observer à la figure 1.4 et à l'équation 1.2.

Un perceptron peut ainsi être vu comme une petite unité de calcul qui peut manipuler des valeurs binaires ou réelles. Les valeurs binaires sont représentées par 0 et 1 ou -1 et 1. Différentes fonctions peuvent être utilisées pour le calcul de la sortie, ce que nous allons voir dans les sections suivantes.

1.2.1 Perceptron multicouche (MLP pour *Multi-layer Perceptron*)

Un perceptron multicouche (MLP) est un réseau de neurones artificiels connecté par des liens pondérés et qui comporte au moins une couche cachée comme l'illustre la figure 1.5. Une couche cachée n'est ni la couche d'entrée ou la couche de sortie et les poids associées à chaque fonction doivent être appris. Son apparition a permis de résoudre un inconvénient majeur de la classification non linéaire.

Ainsi, dans la section précédente, nous avons vu que le perceptron de Rosenblatt (1958) ne peut résoudre un problème de classification où la surface de décision

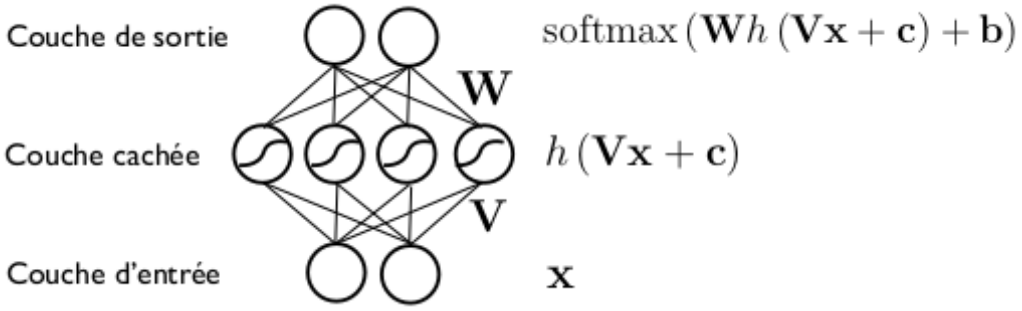

Figure 1.5: Illustration d'un réseau de neurones artificiel extrait de Larochelle (2009)

n'est pas une droite, comme à la figure 1.6 ou les données ne peuvent être séparées linéairement.

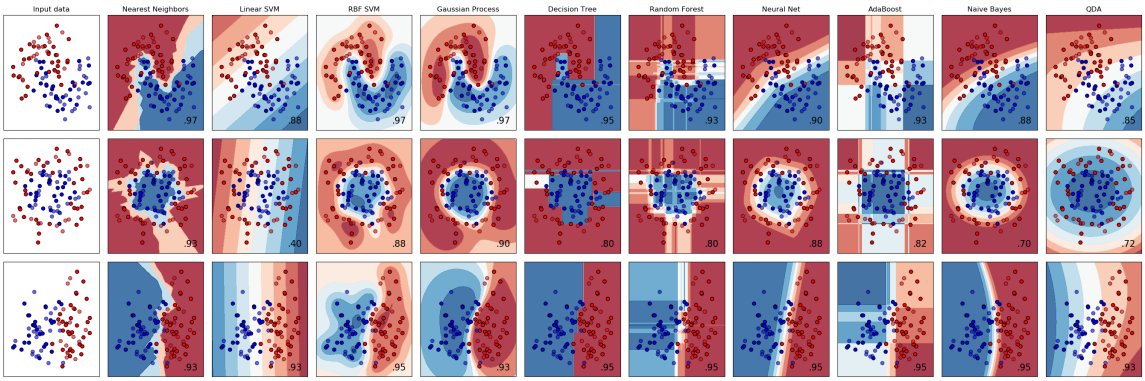

Figure 1.6: Données non linéairement séparables. Extrait de Pedregosa *et al.* (2011)

Ainsi, selon Goodfellow *et al.* (2016), un réseau de neurones est un graphe dirigé acyclique qui décrit comment des fonctions sont liées entre elles. Par exemple, on a trois fonctions, $f^{(1)}$, $f^{(2)}$ et $f^{(3)}$ composées en chaîne pour former $f_{(x)} = f^{(3)}(f^{(2)}(f^{(1)}(x)))$. Donc, dans ce modèle, $f^{(1)}$ est la première couche du réseau, $f^{(2)}$ la deuxième et ainsi de suite. La profondeur de cette chaîne de fonction est ce qu'on appelle la profondeur du modèle et c'est de là, d'ailleurs, que vient le nom apprentissage profond.

Lors de l'entraînement, le réseau tente de trouver une solution en minimisant l'erreur. On fournit à l'algorithme des données annotées, mais on ne spécifie pas quels sont les poids pour chaque neurone ou chaque couche puisque l'algorithme décide de la meilleure approximation de la solution. Les données ne spécifiant pas les valeurs des paramètres des couches, nous les appelons des couches cachées.

Les neurones artificiels sont les unités de calculs du modèle. Ainsi, chaque neurone a une entrée (un scalaire), une fonction d'activation non linéaire différentiable et une sortie. Le nombre de neurones dans les couches est ce qui détermine la taille d'un modèle. Le but d'un MLP est de proposer une approximation de fonction.

Mathématiquement, nous pouvons définir un réseau de neurones, le plus simple possible, à une couche cachée comme ceci :

$$x = f(s) = B\Omega(As + a) + b \tag{1.3}$$

Où s est un vecteur d'entrée et x un vecteur de sortie. A est la matrice de poids et a est le vecteur de biais de la première couche. B est la matrice de poids et b le vecteur de biais de la deuxième couche. Finalement, Ω est la fonction d'activation non linéaire.

Comme démontré par Hornik (1991), un réseau à une couche cachée, comme dans l'équation 1.3, peut approximer n'importe quelle fonction continue telle qu' $f : \mathbb{R}^n \to \mathbb{R}^m$ à n'importe quelle précision arbitraire, s'il y a suffisamment d'unités dans la couche cachée. On qualifie ainsi les réseaux de neurones artificiels d'**approximateurs universels**.

Il nous reste toutefois à explorer tout ce qui entre dans la conception d'un tel modèle. Premièrement, nous allons présenter l'algorithme de **rétropropagation**

de gradient et ses variantes modernes qui permettent d'entraîner le modèle en calculant les gradients en tirant avantage de **la règle de dérivation en chaîne** afin de factoriser les différents éléments de calcul pour l'ensemble des gradients et faire leurs estimations. Ensuite, le MLP introduit aussi le concept de **fonction d'activation** qui est utilisée pour calculer les valeurs de sortie de la couche cachée. Finalement, il y a l'entraînement : il faut choisir un optimiseur, une fonction de coût et la forme que prendra la couche de sortie.

Ceci est l'architecture générale d'un réseau de neurones, mais il en existe plusieurs types, dont notamment les réseaux convolutifs (CNN) et les réseaux récurrents (RNN) que nous allons explorer dans les chapitres suivants. Ainsi, un CNN peut classifier des images complexes et RNN est bon pour traiter des données séquentielles, car il garde en mémoire les états précédents. Il en existe beaucoup d'autres, mais cela dépasse le champ d'études de ce mémoire.

1.2.2 Rétropropagation du gradient (*Backpropagation*)

Il existe deux grandes étapes à l'entraînement d'un MLP. Premièrement, il faut obtenir la valeur prédite, c'est ce qu'on appelle la passe par en avant. Ensuite, c'est au tour de la passe arrière. Elle débute à la couche de sortie ou elle propage le gradient du coût obtenu à la première passe par toutes les couches en utilisant la règle de la **dérivation en chaîne (*Chain rule*)**.

Il faut préciser que l'algorithme de rétropropagation n'est que la partie qui calcule les gradients lors de l'entraînement. La descente de gradient stochastique réalise l'apprentissage. Elle rend donc cet algorithme plus efficace. Toutefois, l'optimisation de l'entraînement d'un réseau est non seulement ardue, mais est aussi un problème d'optimisation non convexe. Ainsi, il n'est pas garanti qu'il existe un seul minimum global. Il est possible qu'il y ait aussi des minimums locaux, comme

illustré par la figure 1.7.

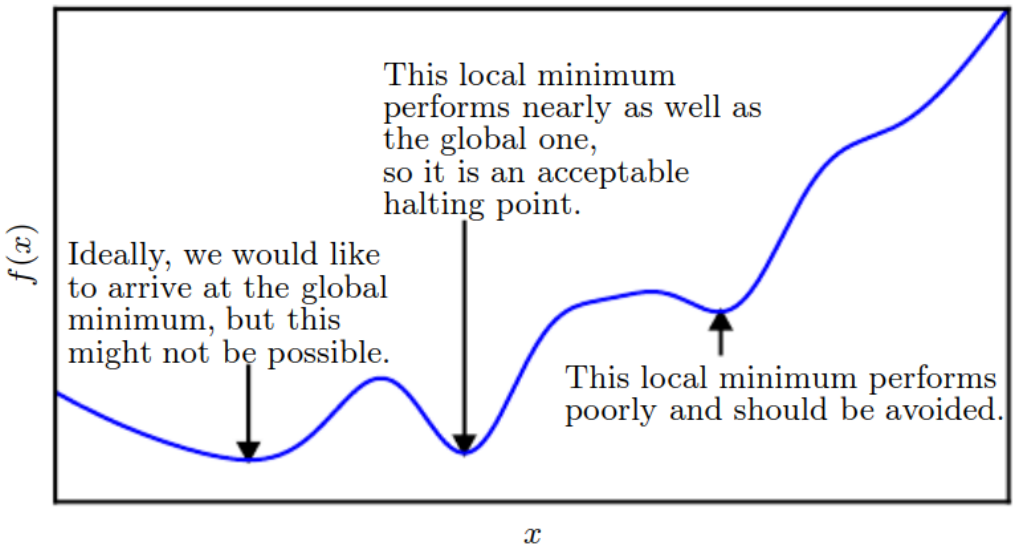

Figure 1.7: Les algorithmes d'optimisation peuvent échouer à trouver un minimum global lorsqu'il y a plusieurs minimums locaux présents. Dans l'apprentissage profond, on accepte généralement de telles solutions même si elles ne sont pas vraiment optimales, car elles correspondent à des valeurs significativement basses de la fonction de coût (Goodfellow *et al.*, 2016)

1.2.3 Fonctions d'activation

Les fonctions d'activation sont des fonctions mathématiques dérivables qui déterminent la sortie d'un réseau de neurones. La fonction s'attache à une unité (un neurone) du réseau et détermine s'il doit être activé ou non. De plus, les fonctions peuvent aider à normaliser les sorties d'un réseau dans un intervalle entre 0 et 1 ou entre -1 et 1.

Un autre aspect très important de ces fonctions est qu'elles doivent être très performantes en complexité mémoire et temporelle. Les réseaux de neurones modernes ont parfois des millions de neurones, ainsi, il est essentiel que ces fonctions soient performantes, mais surtout, ces fonctions doivent être dérivables pour permettre

la descente du gradient.

Les 4 types de fonctions d'activation en apprentissage automatique les plus connues sont la fonction logistique sigmoïde, tangente hyperbolique, ReLU et Leaky ReLU.

La fonction logistique (sigmoïde)

La plus connue et la plus ancienne des fonctions d'activation est la **sigmoïde** (figure 1.8). Elle prend en entrée une valeur réelle et la transforme en une valeur réelle entre 0 et 1.

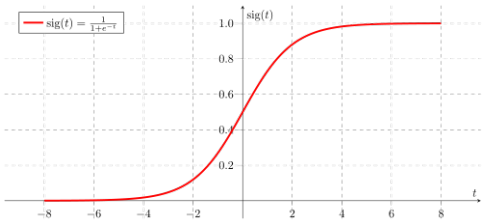

Figure 1.8: Illustration de la fonction logistique sigmoïde extraite de Google (2021). Licence CC.

Toutefois, l'utilisation de la fonction sigmoïde pose certains problèmes : lorsqu'elle traite des valeurs extrêmes, il n'y a presque pas de changement pour les prédictions, c'est-à-dire que le gradient de la fonction est proche de zéro lorsque les valeurs sont trop grandes ou trop petites. Ceci provoque ce qu'on appelle le *vanishing gradient* ou la disparition du gradient, ce qui signifie que le gradient de la fonction coût diminue progressivement vers zéro et que l'apprentissage devient trop lent, rendant l'apprentissage difficile.

De plus, les valeurs de sortie de la fonction sigmoïde ne sont pas centrées à 0. Les données de sortie sont toujours positives ce qui complique l'algorithme de descente du gradient. Ainsi, selon LeCun *et al.* (2012) le gradient sur les poids lors de la

rétropropagation sera toujours positif ou négatif forçant SGD à faire des zigzags et rendant la convergence plus lente. Malgré tout, sigmoïde est pratique lorsqu'on veut obtenir une probabilité puisqu'elle existe en 0 et 1 elle aussi.

Tangente hyperbolique (tanh)

La fonction tangente hyperbolique (figure 1.9), notée tanh est généralement un meilleur choix que la fonction sigmoïde. Elle s'écrit mathématiquement comme suit :

$$\begin{array}{rccc} \tanh : & \mathbb{C} \setminus \mathrm{i}\pi \left(\mathbb{Z} + \dfrac{1}{2} \right) & \longrightarrow & \mathbb{C} \\ & z & \longmapsto & \dfrac{\sinh(z)}{\cosh(z)} \end{array} \tag{1.4}$$

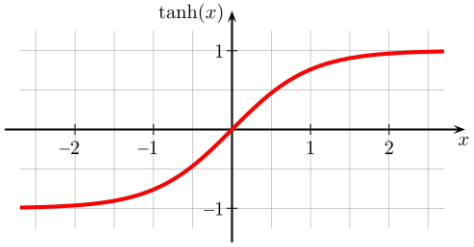

Figure 1.9: Illustration de la tangente hyperbolique (tanh) extraite de Google (2021). Licence CC.

Elle transforme les entrées en nombre réel compris entre -1 et 1, donc elle centre les données à zéro. Toutefois, elle peut aussi provoquer le problème du *vanishing gradient*, car elle tend aussi vers l'infini à ses extrêmes et donc sa dérivée devient nulle.

Rectified Linear Unit (ReLU)

La fonction d'activation la plus populaire de nos jours est sans conteste la *Rectified Linear Unit (ReLU)*, figure 1.10, une fonction linéaire par morceaux.

$$f(x) = x^+ = max(x, 0) \tag{1.5}$$

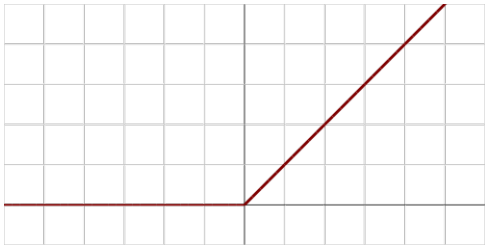

Figure 1.10: Illustration de la fonction logistique (ReLU) extraite de Google (2021). Licence CC.

Son principal avantage est qu'elle est très performante et elle permet au réseau de converger très rapidement Glorot et Bengio (2010); Goodfellow *et al.* (2016). De plus, bien qu'elle soit linéaire par morceaux, ReLu est dérivable et permet la rétropropagation.

Toutefois, elle peut provoquer le problème de *Dying ReLU* (Goodfellow *et al.*, 2016), c'est-à-dire que lorsque les entrées approchent de zéro, le gradient de la fonction devient lui aussi zéro. Ainsi le réseau ne peut plus faire de rétropropagation et ne peut plus apprendre.

Pour pallier au problème du *Dying ReLU*, *Leaky ReLU* change la valeur au-dessous de zéro, en lui ajoutant petite valeur ϵ, l'empêchant ainsi de mourir.

$$f(x) = x^+ = max(x, \epsilon x) \tag{1.6}$$

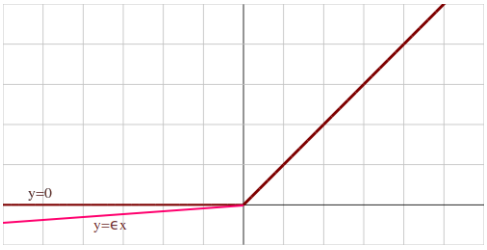

Figure 1.11: Illustration de la fonction logistique (Leaky ReLU) extraite de Google (2021). Licence CC.

Swish

Le nouveau venu, selon Ramachandran *et al.* (2017), est plus performant que *ReLU* et est similaire en matière de complexité de calcul. Dans leurs expérimentations, *Swish* a obtenu de meilleurs résultats de 0,6% sur *Inception-ResNet-v2* en remplaçant ReLU par Swish.

$$f(x) = x * sigmoid(\beta x) \tag{1.7}$$

Swish est en fait une variante de la fonction sigmoïde, mais avec un poids et un paramètre entraînable β. Ainsi swish apporte certains avantages : c'est une fonction continue, il permet la propagation à une petite quantité de poids négatifs. Finalement, le paramètre entraînable β permet un meilleur ajustement de la fonction d'activation pour maximiser la propagation d'information et des gradients plus lisses, ce qui fait que le modèle généralise mieux et plus rapidement (Rama-

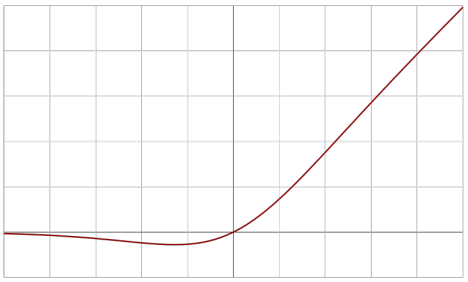

Figure 1.12: Illustration de la fonction d'activation (swish) extraite de Google (2021). Licence CC.

chandran *et al.*, 2017).

Softmax

Softmax (figure 1.13), contrairement aux autres fonctions, est capable de gérer plusieurs classes ou plusieurs sorties. Ainsi, il normalise ses sorties en nombres compris entre 0 et 1. De plus, la somme de ces nombres vaut 1, ce qui en fait une distribution de probabilités (Goodfellow *et al.*, 2016).

Softmax est souvent utilisé dans la dernière couche des classifieurs car il permet de classifier selon une distribution de probabilités.

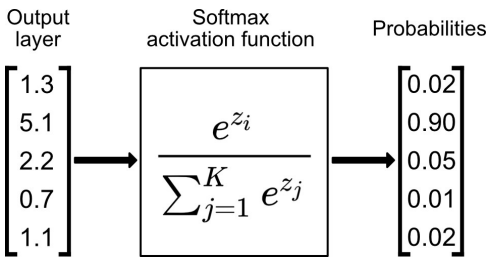

Figure 1.13: Illustration de la fonction softmax extraite de Google (2021). Licence CC.

1.2.4 Réseaux de neurones convolutifs (CNN)

Depuis leur création dans les années 80 grâce aux travaux de Fukushima (1980), les réseaux de neurones à convolution (CNN) ont résolu avec succès de nombreux problèmes tels que la reconnaissance de l'écriture manuscrite dans les années 1990. Ces dernières années, l'apparition de meilleures unités de traitement graphique (GPU) a rendu possible l'architecture de réseaux de neurones convolutifs (figure 1.14) plus profonds qui ont montré une efficacité impressionnante dans la résolution des tâches de reconnaissance d'image. Ces réseaux s'inspirent des travaux de Hubel et Wiesel (1962) sur le système visuel des chats et de leurs propriétés, soit les champs récepteurs locaux et le partage de poids.

Dans ce mémoire, nous portons une attention particulière à ceux-ci, car l'état de l'art les utilise autant dans la détection de texte que dans sa localisation dans l'image. De plus, la reconnaissance de caractère l'utilise pour extraire les caractéristiques du texte à reconnaître qui servira éventuellement d'entrée à un réseau de neurones récurrents.

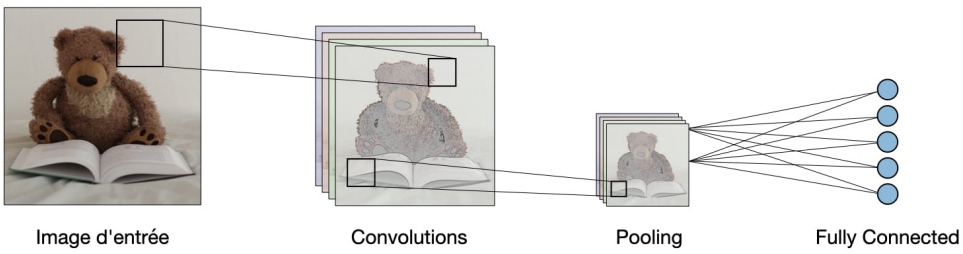

Figure 1.14: Architecture d'un CNN traditionnel extrait de Amidi et Amidi (2021a) Licence CC.

Convolution

La couche de convolution (en anglais convolution layer ou CONV) (figure 1.15) utilise des filtres qui scannent l'entrée ι suivant ses dimensions en effectuant des opérations de convolution. Elle peut être réglée en ajustant la taille du filtre **F** et l'espacement ou *stride* **S**. La sortie **O** de cette opération est appelée carte d'attributs ou *feature map* en anglais.

En d'autres mots, pour une image de 5x5 pixels, un carré ayant un filtre de 9 pixels (3x3) et un espacement de 0, filtre l'image. La sortie est donc une carte d'attributs de 3x3 pixels.

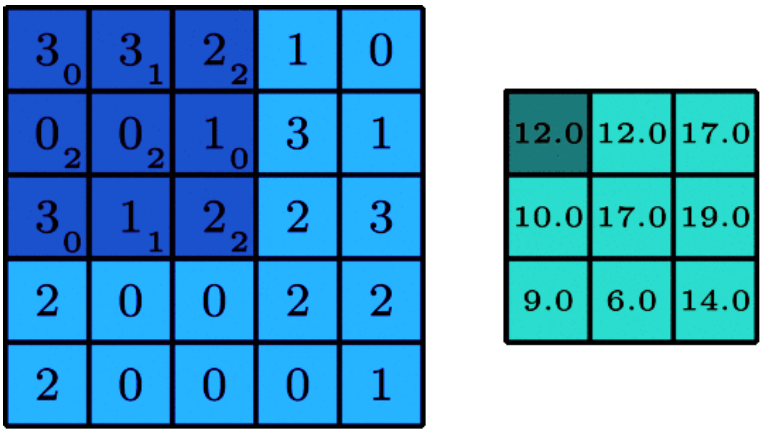

Figure 1.15: Exemple de convolution extrait de Amidi et Amidi (2021a). Licence CC.

Pooling

Les CNN sont principalement constituées de 2 composants, une couche de convolution et une couche de mise en commun ou *pooling* (figure 1.16). La couche de *pooling* sous-échantillonne les résultats des couches convolutives et réduit pro-

gressivement les données sur tout le réseau. Les paramètres CNN peuvent croître de façon exponentielle et la mise en commun aide à les réduire, augmente les performances du réseau et agit comme un régularisateur empêchant le surapprentissage (Goodfellow *et al.*, 2016).

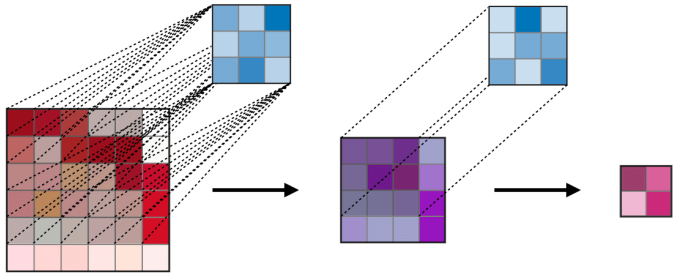

Figure 1.16: Exemple d'une opération de *pooling* de taille 3x3 (en bleu) sur des cartes d'attributs en rouge, extrait de Amidi et Amidi (2021a) Licence CC.

Toutefois, malgré les performances étonnantes des CNN, ils ont encore quelques défauts et c'est ce qui motive la communauté de recherche à toujours essayer de les améliorer. De nombreuses méthodes de mise en commun existent, mais les plus populaires sont le *max pooling* et l'*average pooling*, où les valeurs maximales et moyennes sont prises, respectivement.

Le célèbre pionnier de l'apprentissage automatique Geoffrey Hinton a d'ailleurs déclaré, à propos du maximum pooling : "The pooling operation used in convolutional neural networks is a big mistake and the fact that it works so well is a disaster." (Hinton, 2015). Hinton essaie donc de promouvoir le *Capsule Neural Network* (CapsNet), une variante des CNN où l'on ajoute des capsules qui préservent les informations hiérarchiques, permettant ainsi de garder un certain contexte local et global.

Hinton faisait référence à la perte d'information inévitable avec le *maximum pooling* et le *average pooling*. De plus, le *maximum pooling* a tendance à surapprendre

les données, car il choisit toujours les même grandes valeurs dans les cartes d'attributs. L'*average pooling*, en revanche, est connu pour diluer les caractéristiques et créer un flou sur l'image (Yu *et al.*, 2014).

Au fil des années, de nombreuses autres techniques de mise en commun ont émergé. Williams et Li (2016) introduisent ainsi de nouvelles variantes telles que *pooling* mixte ou stochastique afin de comparer les performances de leur nouvelle approche, appelée le wavelet pooling.

Le *pooling* mixte, comme son nom l'indique, est un mélange de moyenne et de *maximum pooling*. Dans le *stochastic pooling*, l'algorithme choisit aléatoirement quel mise en commun utiliser.

Zeiler et Fergus (2013) affirment que le *stochastic pooling* évite les lacunes du maximum and average pooling, tout en bénéficiant de certains des avantages du *maximum pooling*. En choisissant aléatoirement entre la moyenne et le maximum, il permet de mitiger le surapprentissage tout en ne perdant pas trop d'information utile qui serait diluée avec le *average pooling*. Toutefois, en réalité, le *stochastic pooling* introduit un coût de calcul élevé et est difficile à justifier étant donné la perte de vitesse pour une très petite augmentation des résultats.

Couche entièrement connectée (**FC**)

La couche entièrement connectée (en anglais *fully connected layer*) (**FC**) s'applique sur un vecteur d'entrée obtenu par l'aplatissement d'une matrice où chaque entrée est connectée à tous les neurones. Les couches **FC** (figure 1.17) sont typiquement situées à la fin des architectures de CNN et peuvent être utilisées pour optimiser des objectifs tels que les scores de classe.

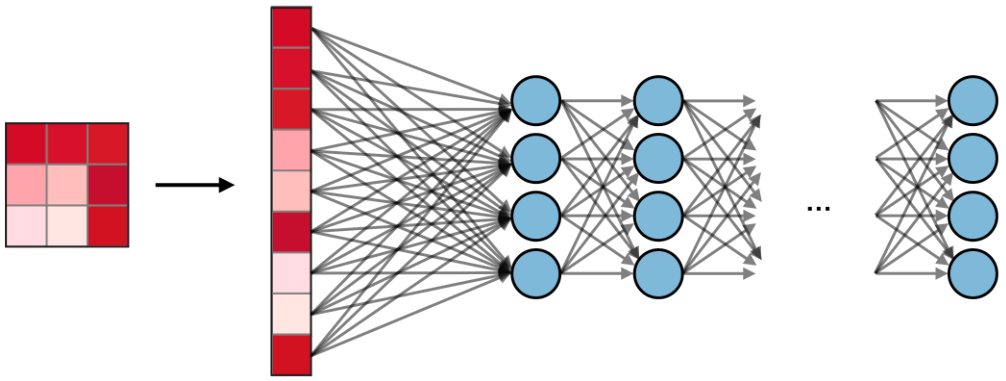

Figure 1.17: Couche entièrement connectée, extrait de Amidi et Amidi (2021a). Licence CC.

1.2.5 Réseaux de neurones récurrents (RNN)

Les réseaux de neurones récurrents (RNN pour *recurrent neural networks*) (Jain et Medsker, 1999), sont une extension des réseaux de neurones qui permettent aux prédictions antérieures d'être réutilisées comme entrées, comme l'illustre la figure 1.18. Ainsi, les RNN peuvent traiter des entrées de longueur variable et sont ainsi très utiles pour la classification de séquence. Cela permet de réaliser des tâches comme la reconnaissance vocale ou de séries temporelles comme la prédiction du cours de la bourse. Les RNN sont aussi essentiels en reconnaissance d'écriture manuscrite et c'est pourquoi ils sont importants dans notre travail de recherche.

Un RNN supporte des tailles variables d'entrées en utilisant des états cachés récurrents. Cela permet de garder des informations sur le contexte en apportant des éléments passés.

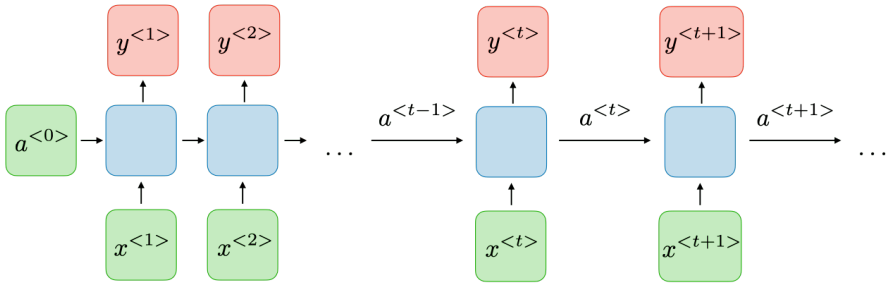

Figure 1.18: Architecture d'un réseau de neurones récurrents. En vert, ce sont les entrées, en bleu, les couches cachées et en rouge la couche de sortie. Extrait de Amidi et Amidi (2021b). Licence CC.

1.3 Traitement automatique du langage naturel (**TALN**)

Le traitement automatique du langage naturel (**TALN**) est un secteur de recherche varié qui mélange linguistique, informatique et intelligence artificielle. Il vise, comme son nom l'indique, à traiter automatiquement le langage pour diverses raisons.

Dans le cadre de la détection et de la reconnaissance de caractères dans les scènes naturelles, le TALN est important pour faire la post-correction du texte, pour traiter les similitudes, pour l'indexation dans un moteur de recherche ou encore pour extraire des informations des mots reconnus.

1.3.1 Modèle de langage

Un modèle de langage est généralement un modèle statistique qui modélise la distribution de séquences de mots.

Historiquement, en TALN, on utilise le **Bag Of Word (BOW)** pour représenter les mots dans un document. Il s'agit d'un vecteur de la taille du vocabulaire

où seuls les mots présents sont marqués avec leur nombre d'occurrences comme l'illustre le tableau 1.1. Cette méthode de représentation est peu efficace, car on perd de l'information sur la relation entre les mots et elle utilise potentiellement trop de mémoire, car plus la taille d'un vocabulaire est grande, plus les vecteurs seront grands.

Document	**Le**	**petit**	**chat**	**colla**	**chien**
le chat colla le chien	2	0	1	1	1
le petit chat colla	1	1	1	1	0
le petit chien colla le petit chat colla	2	2	1	2	1

Tableau 1.1: Bag Of Words

Pour pallier ces problèmes, trois grandes stratégies ont émergé, soit le clustering, la représentation distributionnelle et la représentation distribuée de plongements de mots ou *word embeddings* en anglais. Nous allons les explorer plus en détail dans l'état de l'art 2.5.

1.3.2 Clustering

Comme nous l'avons vu dans la section sur l'apprentissage non supervisé et le partitionnement avec l'algorithme de K-moyenne, le *clustering* consiste à regrouper les mots en s'appuyant sur leur contexte.

1.3.3 La représentation distributionnelle

Ces représentations sont construites grâce à des matrices de co-occurences de mots. Le mot est représenté par un vecteur dans lequel chaque entrée est une mesure d'association entre le mot et un contexte.

1.3.4 La représentation distribuée de plongements de mots

Récemment, Bengio *et al.* (2003) proposent de représenter les mots par un vecteur dense de faibles dimensions. Ainsi, lorsque les vecteurs de mots sont proches en termes de distance, les mots qui les composent doivent aussi l'être sémantiquement.

1.3.5 Indexation

L'indexation est un processus visant à rendre accessibles, plus facilement et rapidement, des informations. De façon analogue à un bottin téléphonique où l'on peut chercher facilement un numéro de téléphone, on utilise un index pour retrouver des documents.

L'indexation automatique de documents permet d'organiser des collections de documents en faisant usage de l'extraction de caractéristiques, du partitionnement des données, de la recherche d'informations.

Stemmatiseur

Le stemmatiseur recherche selon la forme du mot et de sa langue, le radical le plus probable pour ce mot. À l'inverse de la lemmatisation, qui repose sur une base de connaissance de la langue auxquelles on associe les lemmes possibles (appelée lexique), la stemmatisation fonctionne uniquement avec des règles syntaxiques et grammaticales (Manning et Schütze, 1999). Elle constitue d'ailleurs une de ses forces puisqu'elle est plus robuste aux fautes d'orthographe.

N-gramme de Porter (1980) : est une sous-séquence de n éléments construite à partir d'une séquence donnée. En d'autres termes, elle représente une sous sé-

quence d'un autre mot. Par exemple *mém* est un trigramme de mémoire et *moir* une quadrigramme. Ainsi, n-gramme permet de créer un modèle probabiliste qui anticipe le prochain élément d'une suite ou encore de représenter un mot selon toutes les possibilités de sous séquence choisit. Par exemple, si on choisit la taille minimale à 1 et maximale à 2, on obtient pour les mots *Quick Fox* [Q, Qu, u, ui, i, ic, c, ck, k, "k ", " ", " F", F, Fo, o, ox, x].

Stemmatiseur Porter (1980) : Le plus célèbre stemmeur Porter est un algorithme de racinisation/désuffixation qui utilise des heuristiques et des phases pour extraire des mots racines en anglais. Comme l'illustre la figure 1, chaque étape extrait la racine et/ou un suffixe pour extraire l'essentiel d'un mot.

Algorithm 1 Exemple de l'algorithme Porter sur le mot *Generalization*

1: étape 1 : Generalization
2: étape 2 : Generalize
3: étape 3 : General
4: étape 4 : Gener

Stemmatiseur Snowball (Porter2) : Une amélioration du célèbre stemmatiseur Porter (Porter, 1980). Il est légèrement plus rapide et légèrement plus performant grâce à l'addition d'heuristiques. Sur le site de Lucene (http ://lucene.apache.org), ils décrivent le StandardAnalyser : *"This is the most sophisticated analyzer and is capable of handling names, email addresses, etc. It lowercases each token and removes common words and punctuations, if any."*

Stemmatiseur Simple Analyser : Ici, on effectue le traitement minimal, c'est-à-dire qu'on partitionne le texte en termes et qu'on le transforme en minuscule.

Ainsi, lorsqu'on parle de Porter2 ou Snowball dans cette recherche, on fait référence à un ensemble d'analyseurs qui partitionne le texte et le nettoie en enlevant les mots vides ou *stop words* en anglais, les caractères HTML ou encore transforme le texte en ASCII.

Term frequency–inverse document frequency (TF-IDF) : Une méthode de pondération souvent utilisée en recherche d'information (RI). Elle permet d'évaluer l'importance d'un terme (sa fréquence) dans un document par rapport à sa collection ou à un corpus.

Il existe plusieurs mesures de fréquence pour un terme, mais la plus simple est de définir $tf_{i,d}$ à 1 si le terme est dans le document et 0 sinon.

Ensuite, on calcule la fréquence inverse du document pour obtenir l'importance du terme dans l'ensemble du corpus.

$$idf_i = \log \frac{|D|}{|\{d_j : t_i \in d_j\}|} \tag{1.8}$$

où :

— |D| : nombre total de documents dans le corpus ;
— $|\{d_j : t_i \in d_j\}|$: nombre de documents où le terme t_i apparaît (c'est-à-dire $n_{i,j} \neq 0$).

Finalement, on obtient le résultat en multipliant les deux résultats.

$$tfidf_{i,d} = tf_{i,d} \cdot idf_i \tag{1.9}$$

Okapi Best match 25 (BM 25) (Robertson et Zaragoza, 2009) : Une méthode de pondération souvent utilisée en RI, de type TF-IDF qui classe un ensemble de documents en fonction des termes d'une requête apparaissant dans chaque document, quelle que soit leur proximité dans le document.

Similarité cosinus : Calcule la similarité entre 2 vecteurs. Dans notre étude, elle est utilisée pour rechercher la similitude entre 2 vecteurs de plongements de mots

extraits lors de la reconnaissance de texte.

$$cos(\boldsymbol{x}, \boldsymbol{y}) = \frac{\boldsymbol{x} \cdot \boldsymbol{y}}{||\boldsymbol{x}|| \cdot ||\boldsymbol{y}||} \tag{1.10}$$

où :

x et y sont des vecteurs de plongement de mots

1.3.6 Discussion

Ainsi, nous faisons ici un survol de plusieurs notions de traitement automatique du langage, car elles jouent un grand rôle dans ce mémoire en rendant plus robustes les données extraites des films. Le TALN pourra permettre entre autres d'extraire des représentations de mots de la reconnaissance de texte dans les films, de filtrer les informations inutiles, de corriger le texte extrait, de joindre par similitude le texte extrait dans des séquences d'images, d'indexer ce texte avec le temps où il apparaît dans le film, et finalement, cela permet de faire des recherches dans la collection, grâce à un moteur de recherche, et de trouver les informations et de les trier par pertinence.

CHAPITRE II

ÉTAT DE L'ART

Historiquement, la reconnaissance de texte dans les images et vidéos fut séparée en plusieurs tâches distinctes : l'extraction d'images, la transformation d'image, la détection de contours, la détection de texte, la reconnaissance de texte, la correction de texte, etc. Toutefois, comme le problème de reconnaissance de texte dans les documents (OCR) était pratiquement résolu (Gómez et Karatzas, 2015), il semblait normal de se concentrer uniquement sur les étapes préliminaires de celle-ci, soit la détection et la segmentation de texte.

Il faut ajouter que les méthodes de détection de texte s'inspirent largement de la détection d'objets. Ainsi, il est nécessaire d'en faire un survol pour aborder l'état de l'art de la détection de texte.

D'ailleurs, les méthodes modernes font simultanément la détection et la segmentation, ce qui permet non seulement de détecter où est le texte, mais d'en extraire les coordonnées dans l'image pour les traiter par la suite.

2.1 Évaluation

Dans le cadre de ce mémoire, nous utilisons plusieurs métriques pour analyser les résultats et dans cette section nous en faisons un survol.

La **précision** mesure le ratio de bons documents retrouvés par rapport à tous les documents retrouvés :

$$précision = \frac{vrais\_positifs}{vrais\_positifs + faux\_positifs} \tag{2.2}$$

Le **rappel** mesure le ratio de bons documents retrouvés par rapport à tous les documents pertinents (*c.-à-d.* qu'il aurait fallu retrouver) :

$$rappel = \frac{vrais\_positifs}{vrais\_positifs + faux\_negatifs} \tag{2.3}$$

La **F-mesure** est la moyenne harmonique de la précision et du rappel. Elle offre une meilleure mesure plus balancée.

$$F_m = \frac{2 * précision * rappel}{rappel + précision} \tag{2.4}$$

La **mesure de précision moyenne (mAP)** est, pour un ensemble de requêtes, la moyenne des scores de précision pour chaque requête.

$$mAP = \frac{\sum_{q=1}^{Q} précision(q)}{Q} \tag{2.4}$$

où Q est le nombre de requêtes.

Tests d'union-intersection (IoU pour *Intersection over Union*) Padilla *et al.* (2020) : En détection d'objets, les jeux de données fournissent des coordonnées qui localisent les objets dans les images. Or les prédictions ont rarement des coordonnées identiques, ce qui ne veut pas dire qu'elles sont mauvaises. L'important est que l'objet soit bien localisé et que les coordonnées soient proches de celles fournies par le jeu de données.

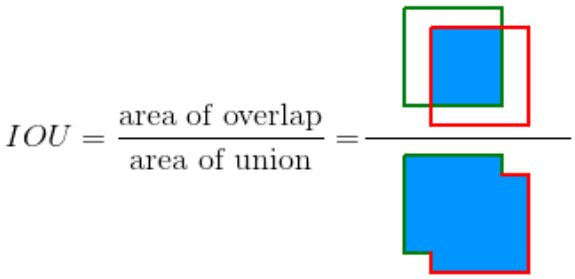

Figure 2.1: Tests d'union-intersection extrait de Padilla *et al.* (2020)

Nous avons donc besoin d'une mesure relative qui maximise l'IoU entre la prédiction et les vraies données comme l'illustre la figure 2.1.

Dans la compétition de localisation de texte de l'ICDAR, 2 protocoles d'évaluation sont utilisés, the DetEval (Wolf et Jolion, 2006) et PASCAL (Karatzas *et al.*, 2013b).

DetEval de Wolf et Jolion (2006) est en fait un protocole qui se base sur le nombre de régions qui se chevauchent entre les régions réelles du jeu de données et les régions détectées. Il prend en compte le chevauchement de plusieurs détections sur valeur réelle, et inversement, il prend en compte le chevauchement de valeurs réelles dans une région détecté. Ensuite, il utilise un seuil qui utilise la précision et le rappel pour joindre la prédiction à la donnée réelle comme l'illustre la figure 2.2.

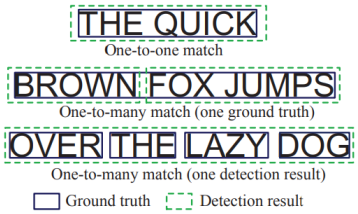

Figure 2.2: Protocole DetEval extrait de Wolf et Jolion (2006)

2.2 Détection d'objets

Dans cette partie, nous allons présenter les dernières avancées dans la reconnaissance d'objets dans les images. Nous avons déjà fait un tour d'horizon des CNN, nous allons donc aborder la reconnaissance et la détection dans les images, car les méthodes modernes le font simultanément. Les modèles les plus importants de l'état de l'art sont présentés ci-après.

R-CNN (Girshick et al.*, 2013)*

Dans leur recherche, Girshick *et al.* (2013) proposent d'utiliser un système en 3 étapes (figure 2.3). La première génère 2000 *propositions de régions*, qui sont des candidats d'objets potentiels pour le détecteur. La deuxième utilise un réseau CNN qui doit extraire les cartes d'attributs (*feature map*) des régions trouvées. Ces caractéristiques sont le résultat des filtres de convolution du CNN, comme nous l'avons expliqué au chapitre 1, et permettent de décrire une image par ses composantes, comme les lignes verticales, horizontales, les courbes, les coins, etc. La troisième étape s'appuie sur un séparateur à vaste marge (SVM) (Cortes et Vapnik, 1995) pour classifier chaque région selon les classes d'objets du jeu de données. Finalement, un algorithme de *Greedy Max Suppression* fait le tri et extrait les régions ayant obtenu le meilleur score.

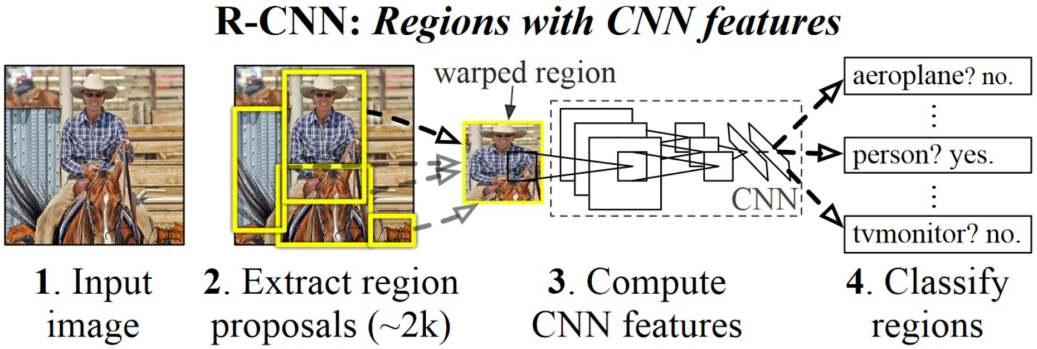

Figure 2.3: Architecture d'un *R-CNN* extrait de Girshick *et al.* (2013)

Ce modèle obtient une précision moyenne (mAP) de 53,7% dans la compétition PASCAL VOC Everingham *et al.* (2009), en 2010. Toutefois, ce modèle à quelques inconvénients, notamment, il est lent et prend beaucoup de mémoire à cause de l'utilisation du CNN pour obtenir les 4096 caractéristiques des 2000 propositions de régions.

Fast R-CNN (Girshick, 2015)
Dans cet article du même auteur que R-CNN, Girshick (2015) proposent d'utiliser un CNN qui prend en entrée une image et qui génère une carte d'attributs (*feature map*). De ces caractéristiques, Fast R-CNN identifie des propositions de régions et les envois à une couche de *pooling* qui les transforme dans une matrice de taille fixe, ceci étant fait, on peut maintenant connecter le réseau de bout en bout et utiliser une couche *Softmax* pour prédire la classe de chaque région.

Ce modèle obtient 66% de mesure de précision moyenne (mAP) à la compétition PASCAL VOC (Everingham *et al.*, 2009). De plus, contrairement à son ancêtre qui avait 3 modules, on peut l'entraîner en une seule étape de bout en bout. Finalement, il est beaucoup plus rapide, car nul besoin de générer 2000 propositions de régions avec un CNN.

Faster R-CNN (Ren et al.*, 2015)*
Contrairement à R-CNN et Fast R-CNN, l'algorithme Faster R-CNN (figure 2.4) n'utilise pas d'algorithme *selective search* (Uijlings *et al.*, 2013) pour générer les propositions de régions. Cet algorithme est très lent et ralentit la performance du réseau.

Ainsi, Faster R-CNN utilise un CNN pour extraire des cartes d'attributs et un réseau est utilisé pour prédire des propositions de régions. Finalement, les cartes d'attributs et les régions sont transmises au classifieur à l'aide d'une couche de pooling. Une fois la classification terminée, l'algorithme de *Non-maximum Sup-*

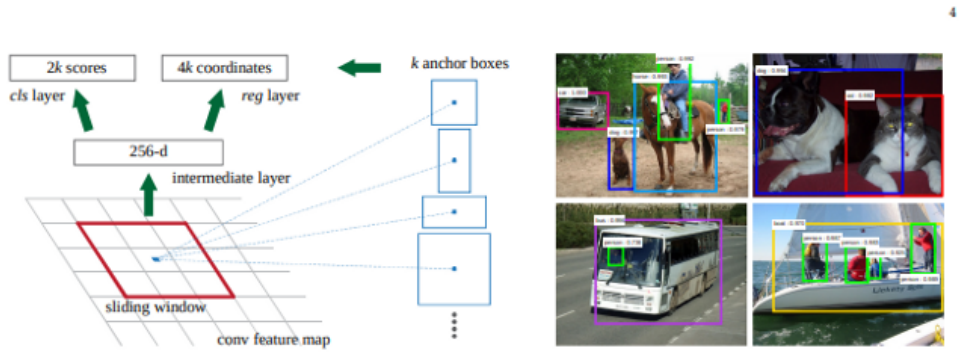

Figure 3: **Left**: Region Proposal Network (RPN). **Right**: Example detections using RPN proposals on PASCAL VOC 2007 test. Our method detects objects in a wide range of scales and aspect ratios.

Figure 2.4: Architecture du *Faster R-CNN* Ren *et al.* (2015)

pression (NMS) (Rosenfeld et Thurston, 1971) élague les détections pour extraire celles qui ont le plus haut mAP. Le gain en performance est significatif, R-CNN traite une image par 50 secondes, Fast R-CNN traite une image par 2 secondes et Faster R-CNN une image par 0.2 seconde.

Single Shot MultiBox Detector - SSD (Liu et al.*, 2015)*
Cet article présente un modèle CNN à une passe qui n'utilise qu'un seul réseau de neurones. Fin 2016, il bat de nouveaux records en obtenant 74,3% de mAP sur PASCAL VOC 2007 Everingham *et al.* (2009), et ce, avec une vitesse de traitement de 59 images par seconde (fps).

SSD est *Single shot* car il permet de détecter et de classifier en une propagation directe (*forward pass* en anglais). Il est *multibox*, ce qui signifie qu'il utilise des boîtes de dimensions prédéfinies sur les cartes d'attributs. Ces boîtes sont choisies soigneusement pour des objets dans les images (figure 2.5). Ainsi, pour chaque taille prédéfinie de carte d'attributs, SSD prédit la classe et les coordonnées de la région dans l'image. Finalement, un algorithme de NMS élague les détections pour extraire celles qui ont la meilleure précision et filtre les doublons.

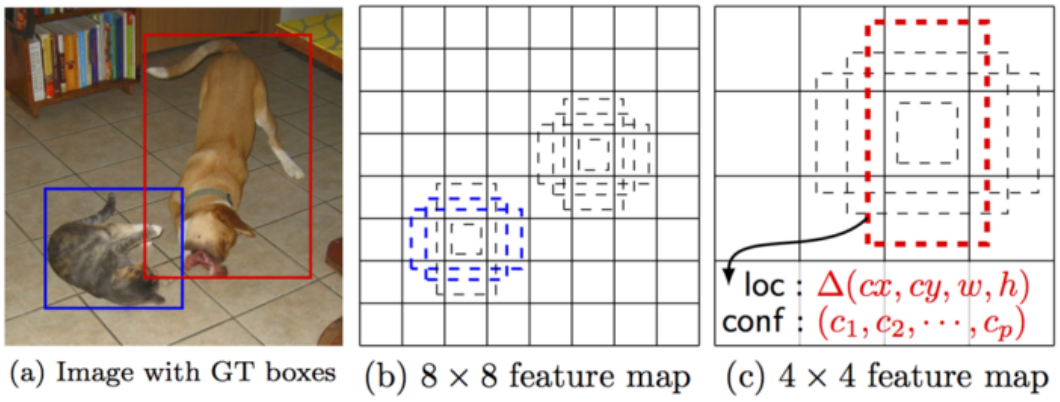

Figure 2.5: Architecture d'un*Single Shot MultiBox Detector*. Extrait de Liu *et al.* (2015)

Pour conclure, SSD-500 (images de résolution 512x512 pixels) obtient 76,8% de mesure de précision moyenne (mAP) sur PASCAL VOC 2007 (Everingham *et al.*, 2009) avec 22 images par seconde (fps). Toutefois, si on descend la résolution à 300 pixels, on obtient 74,3% à 59 fps. Ce qui est un bien meilleur temps de traitement pour une perte de précision négligeable. SSD a malgré tout un grand défaut : il a beaucoup de difficulté à détecter les petits objets ou encore les longs objets rectangulaires, car ils n'apparaissent pas dans les *filtres de convolution* qui ont des dimensions prédéfinies.

YOLO : You Only Look Once (Redmon et al.*, 2015)*

Probablement l'algorithme le plus rapide et le plus populaire présentement, YOLO est rendu à sa cinquième version en 2021. Contrairement aux autres réseaux, YOLO n'utilise pas de proposition de régions, mais est en quelque sorte un gros réseau de neurones convolutif. Il pré sépare ou découpe l'image en plusieurs régions et classifie chacune de ces régions obtenant la classe ainsi que les coordonnées de l'objet. Finalement, YOLO, comme les autres détecteurs, utilise l'algorithme NMS pour filtrer les meilleurs résultats et les doublons.

Dans sa première version en 2016, YOLO permettait de faire de la détection

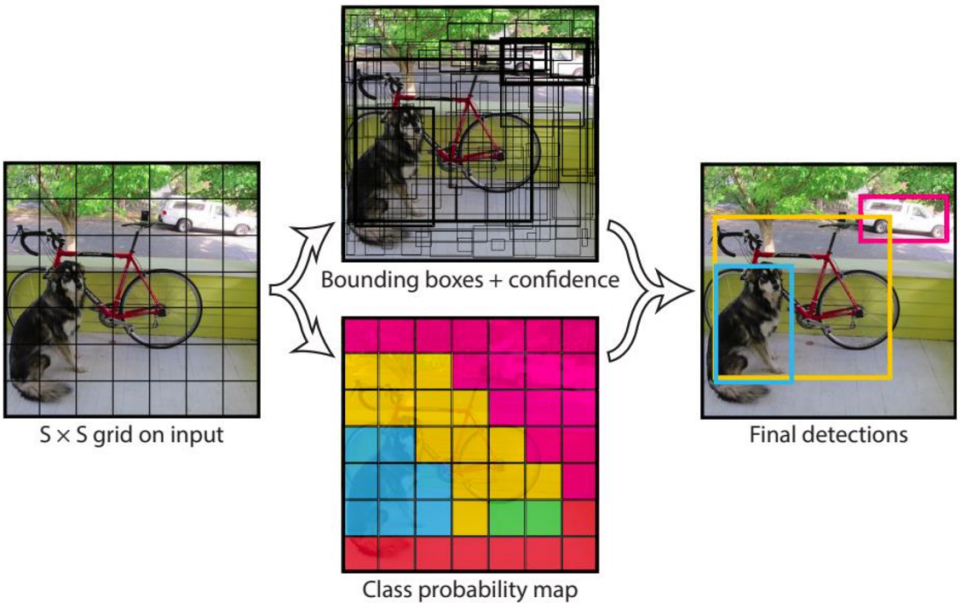

Figure 2.6: Architecture du modèle *YOLO*. Extrait de Redmon *et al.* (2015). Ici YOLO divise l'image en matrice de taille SxS et prédit β régions, leur confiance et leur classe pour chaque cellule.

d'objet en temps réel à 45 fps, mais il comportait un inconvénient majeur : une cellule n'avait qu'une classe, donc, YOLO ne pouvait pas détecter les objets qui se chevauchaient dans l'image (figure 2.6).

En 2017, Redmon et Farhadi (2016) publient YOLO-9000. Ils y ajoutent la normalisation de lot ou *batch normalization* en anglais, obtenant ainsi une amélioration du mAP de 2% sur ImageNet (Russakovsky *et al.*, 2014). Ils augmentent aussi la résolution des images en entrée et YOLO peut maintenant détecter plus de 9000 classes. Pour ce faire, Redmon et Farhadi (2016) joint le jeu de données COCO (Lin *et al.*, 2014) et Imagenet en utilisant un arbre d'ontologie de mots WordNet (Miller *et al.*, 1990).

En 2018, Redmon et Farhadi (2018) sortent encore une nouvelle version, *YOLOv3 : An Incremental Improvement*. Le réseau CNN double de taille, ce qui améliore

grandement le nombre de cartes d'attributs générées par le CNN. Ils améliorent aussi la détection des petits objets en concaténant les couches du CNN ensemble. Les couches au début sont responsables de détecter les grands objets et les couches à la fin les plus petits.

De plus, YOLOv3 abandonne l'algorithme *softmax* pour utiliser la régression logistique. La raison est que *softmax* prend la classe ayant obtenu la meilleure probabilité. Mais qu'en est-il si le jeu de données à 2 classes, dans une même région, tel qu'**Animal** et **Chien** ? Ainsi, YOLOv3 peut maintenant détecter des objets ayant plusieurs classes, pour autant que la probabilité soit au-dessus d'un certain seuil.

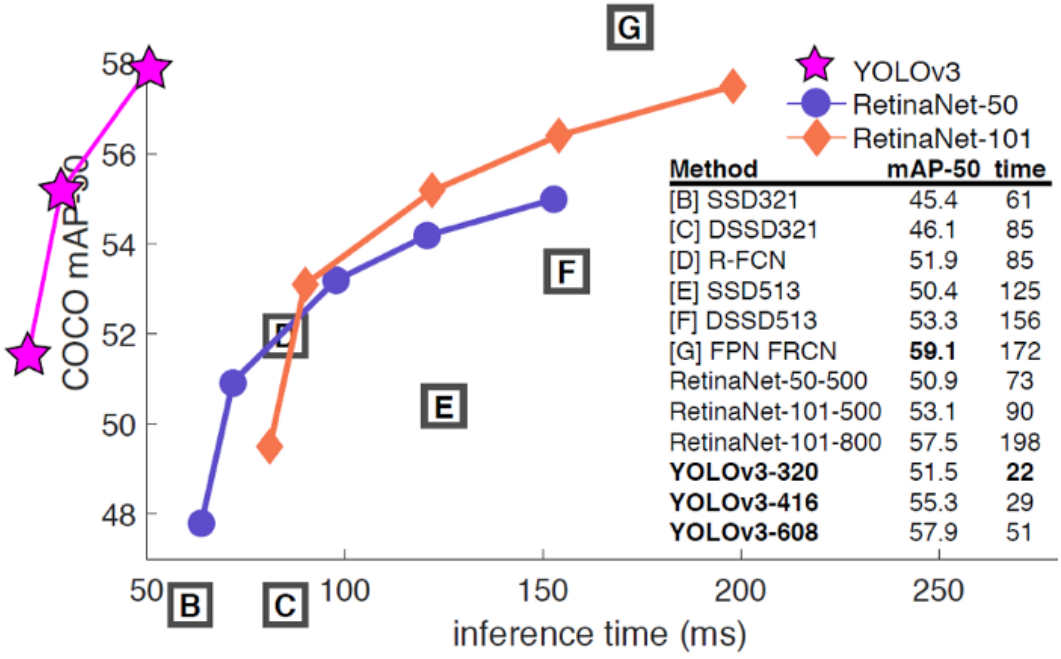

Figure 2.7: Comparaison de la vitesse de calcul de *YOLOv3* par rapport à l'état de l'art. Extrait de Redmon et Farhadi (2018)

Comme l'illustre la figure 2.7, YOLOv3 est considérablement plus rapide que les autres modèles. Dans la figure, il est mentionné Retina Net qui est un nouveau modèle de la famille des détecteurs à une passe (Lin *et al.*, 2017).

Tableau 2.1: Comparaison des différents détecteurs d'objets sur les MS COCO and PASCAL VOC 2012 (*). En gris, ce sont des les détecteurs en temps réel (>30fps).

Model	Year	Size	$AP_{[0.5:0.95]}$	$AP_{0.5}$	FPS
R-CNN*	2014	224	-	58.50%	∼0.02
SPP-Net*	2015	Variable	-	59.20%	∼0.23
Fast R-CNN*	2015	Variable	-	65.70%	∼0.43
Faster R-CNN*	2016	600	-	67.00%	5
R-FCN	2016	600	31.50%	53.20%	∼3
FPN	2017	800	36.20%	59.10%	5
Mask R-CNN	2018	800	39.80%	62.30%	5
DetectoRS	2020	1333	53.30%	71.60%	∼4
YOLO*	2015	(Modifie 448	-	57.90%	45
SSD	2016	300	23.20%	41.20%	46
YOLOv2	2016	352	21.60%	44.00%	81
RetinaNet	2018	400	31.90%	49.50%	12
YOLOv3	2018	320	28.20%	51.50%	45
CenterNet	2019	512	42.10%	61.10%	7.8
EfficientDet-D2	2020	768	43.00%	62.30%	41.7
YOLOv4	2020	512	43.00%	64.90%	31
Swin-L	2021	-	57.70%	-	-

Finalement, en 2020, Bochkovskiy *et al.* (2020) introduisent des YOLOv4 qui augmentent considérablement les résultats de détections sur MS COCO Lin *et al.* (2014), 10% de mAP et 12% de fps. Quelques jours après, Glenn Jocher publie YOLOv5 (Jocher, 2020), créant ainsi un peu de controverse, car le nom n'appartient plus aux auteurs originaux. Ainsi, Glenn reproduit le modèle en PyTorch (Wallach *et al.*, 2019) et y ajoute des améliorations le rendant encore plus performant.

Le paragraphe suivant présente une synthèse qui compare les différents détecteurs d'objets sur MS COCO and PASCAL VOC 2012 pris de Zaidi *et al.* (2021)

2.3 Détection de texte

Il existe deux catégories de méthodes pour la détection du texte dans les images, soit les méthodes à fenêtres glissantes (*sliding windows* en anglais) et les méthodes par composantes connexes (*connected components* en anglais). Les méthodes liées aux réseaux de neurones convolutifs se classent généralement dans la catégorie des fenêtres glissantes (Neumann, 2017).

2.3.1 Composantes connexes

Automatic Text Recognition for Video Indexing (Lienhart, 1997)
Cet article présente l'une des premières recherches en détection de texte dans les vidéos. Lienhart (1997) propose un algorithme en 3 temps pour extraire le texte des films. À la première étape, il regroupe le texte à l'aide de l'algorithme de segmentation *Split and Merge* (Horowitz et Pavlidis, 1976) dans le but de produire image ou le texte est en blanc et le reste en noire. Ensuite, il segmente l'image avec l'analyse des mouvements du texte, car, normalement, il ne devrait pas bouger dans l'image ou encore il ne devrait bouger que linéairement, comme pour des crédits de films par exemple. Finalement, il utilise des algorithmes de transformations morphologiques tels que la dilatation et l'érosion des contours pour faire ressortir le texte pertinent.

Lienhart (1997) prétend obtenir plus de 76% en précision, sur des données annotées par l'auteur (20 vidéos de la télévision allemande) en 1996. Mais lors de récentes compétitions, les solutions n'atteignent pas ce niveau de précision. Donc, nous émettons quelques réserves quant à la validité des résultats dans un contexte plus général que celui des expériences de Lienhart (1997). Toutefois, l'idée de lier le texte dans le temps est intéressante et mérite d'être approfondie avec des techniques modernes.

Real-time scene text localization and recognition (Neumann et Matas, 2012)
Neumann et Matas (2012) proposent d'améliorer les techniques existantes de détection de texte de type grâce à *MSER : Maximally stable extremal regions* (Matas *et al.*, 2004). Rappelons que MSER est utilisé pour détecter des formes floues (ou blob en anglais) dans les images. Intuitivement, on peut expliquer l'algorithme comme ceci : 1) Faire varier les seuils de gris d'une image de façon consécutive et bâtir un arbre de composante. 2) Chercher les formes qui restent stables par rapport aux écarts de seuils. Les auteurs proposent d'utiliser une variation de MSER *CSER : Class-specific extremal regions* qui se distingue en utilisant 2 étapes. Premièrement, un arbre des composants est bâti sans seuil maximal, contrairement à MSER. L'arbre des composants est composé de toutes les formes détectées selon les seuils de gris dans une image. Dans la deuxième étape, on classifie chaque région, en texte/non texte, grâce à un arbre de décision AdaBoost (Schapire, 2013) qui découpe les composants en plusieurs sorties : points d'inflexions de la forme, nombre de trous dans la forme, etc. Ensuite, les sorties sont combinées en une somme pondérée qui représente la sortie finale du classifieur. Enfin, on regroupe les régions classées comme étant du texte en utilisant les caractéristiques des paragraphes, des mots et des lignes, permettant ainsi de nettoyer et de rassembler les régions détectées.

Neumann et Matas (2012) obtiennent, au moment de leur publication, de meilleurs résultats que les gagnants de la compétition ICDAR 2011 (El Abed *et al.*, 2011) pour la reconnaissance de bout en bout. Toutefois, leur approche est très lente, soit à 0,3 seconde par image ayant une résolution de 800x600 pixels. Cette méthode prendrait plus de 7 heures pour traiter un film d'une heure à 24 images par seconde. De plus, avec une F-Mesure d'environ 36,5% pour la reconnaissance de bout en bout, les résultats ne sont pas assez fiables pour être utiles.

Scene Text Recognition : No Country for Old Men ? (Gómez et Karatzas, 2015)

Gómez et Karatzas (2015) proposent de comparer les techniques de reconnaissance de bout en bout avec des OCR normaux tels que ABBYY Finereader (`http://www.abbyy.com`) et Tesseract (Kay, 2007). Ainsi, ils obtiennent un meilleur résultat que Neumann et Matas (2012) en remplaçant leur OCR par une solution commerciale, soit 47,2% de F-Mesure avec ABBY et 40,2% avec Tesseract comparativement à 36,5% sur le jeu de données ICDAR 2011 (El Abed *et al.*, 2011).

Toutefois, les temps d'exécution sont encore plus longs de cette façon, soit en moyenne 851 ms par image. Un film d'une durée d'une heure prendrait 20 heures à traiter. Les auteurs notent que les solutions d'OCR ne supportent que des images en noir et blanc. Il est donc impossible de traiter certaines images dans ces conditions. Cependant, le fait que les auteurs publient le code de leur expérimentation rend cette recherche encore plus intéressante.

2.3.2 Fenêtres glissantes

Text Detection and Character Recognition in Scene Images with Unsupervised Feature Learning (Coates *et al.*, 2011)
Coates *et al.* (2011) proposent d'utiliser des techniques d'apprentissage automatique non supervisé pour la détection et pour la reconnaissance. Ils utilisent un SVM (Cortes et Vapnik, 1995) qui décide si une fenêtre glissante est du texte ou non. Les auteurs utilisent un variant de l'algorithme de partitionnement en k-moyennes Lloyd (1982) pour générer des cartes d'attributs. Ensuite, le système réduit les caractéristiques en utilisant du *pooling* comme dans un CNN et finalement, le SVM détecte la présence de texte.

La fenêtre glissante mesure 32x32 pixels, de sorte que si un caractère est très grand dans l'image ou encore très petit, il ne sera pas détecté. Toutefois, le partitionnement en k-moyennes est très rapide et cette technique pourrait être utile dans

un problème respectant les contraintes de la fenêtre glissante, comme la lecture de plaque d'immatriculation ou de carte de crédit. Les auteurs ne donnent pas de métriques sur leur solution, mais ils mentionnent qu'elle est significativement moins précise que l'état de l'art.

2.3.3 Réseau neuronal convolutif

Reading Text in the Wild with Convolutional Neural Networks (Jaderberg *et al.*, 2014a)
Jaderberg *et al.* (2014a) proposent un système de reconnaissance de bout en bout de texte dans les images. Lors de la détection, les auteurs utilisent deux méthodes empruntées à la détection d'objets, soit une méthode de proposition de région basée sur *Edge Boxes* (Zitnick et Dollár, 2014) et une basée sur *Fast feature pyramids* (Dollar *et al.*, 2014).

Cette recherche de grande qualité s'apparente à notre sujet de recherche. Toutefois, les travaux ont été présentés en 2014 avec une F-mesure de 76% et plus de 9.5 secondes par image sur le jeu de données ICDAR 2013 (Karatzas *et al.*, 2013a). Depuis, plusieurs autres travaux ont obtenu de bien meilleurs résultats.

Synthetic Data for Text Localisation in Natural Images (Gupta *et al.*, 2016)
Gupta *et al.* (2016) proposent deux contributions, soit la création de données synthétiques grâce à une méthode pour générer du texte dans les images et un *Fully-Convolutional Regression Network (FCRN)* qui permet non seulement de trouver des régions de texte, mais aussi de fournir leurs coordonnées. À noter que FCRN est une forme de réseau neuronal convolutif. Les chercheurs s'inspirent de YOLO Redmon *et al.* (2015) puisqu'il supporte les coordonnées, contrairement au FCN classique.

On commence à voir une tendance où les recherches en détection et reconnaissance

de texte s'inspirent des approches utilisées pour les objets. Quant aux résultats, ils suivent eux aussi la progression de la détection d'objets et dans ce cas-ci, surpassent une fois de plus les recherches précédentes. Finalement, la solution est très rapide avec le traitement de 15 images par secondes, rendant possible la détection et la reconnaissance en temps presque réel.

TextBoxes : A Fast Text Detector with a Single Deep Neural Network (Liao *et al.*, 2017)

Liao *et al.* (2017) de façon analogue à Gupta *et al.* (2016), s'inspirent des dernières avancées en reconnaissance d'objet, soit le *Single Shot MultiBox Detector - SSD* (Liu *et al.*, 2015). La grande différence se retrouve dans le fait que *TextBoxes* remplace les couches de SSD par des couches *text-boxes* ayant des rapports de formes (ou aspect ratio en anglais) très larges. Un rapport de forme en télévision est simplement rapport largeur/hauteur, par exemple un film 16/9. Ces nouvelles boîtes permettent ainsi d'obtenir de meilleurs résultats dans la détection de texte par rapport aux objets, car les mots ont souvent des formes rectangulaires.

En 2017, cette approche a obtenu les meilleurs résultats sur les jeux de données ICDAR 2013 (Karatzas *et al.*, 2013a) et ICDAR 2011 (El Abed *et al.*, 2011). Elle permet un traitement nettement plus rapide que les autres méthodes avec 0.73 seconde par images. Toutefois, on remarque qu'un autre système est utilisé pour l'OCR et celui-ci ne détecte que le texte horizontal. Comme plusieurs autres recherches, le code est fourni et libre de droits.

Detecting Oriented Text in Natural Images by Linking Segments (Seglink) (Shi *et al.*, 2017)

Seglink est un modèle de détection de texte orienté. L'idée principale de cette méthode est de décomposer le texte en segments et en liens. Un segment est une boîte d'un mot ou d'une zone de texte et les liens lient les boîtes de texte adjacentes

ensemble.

Le problème avec les détecteurs tels que SSD ou RCNN, est qu'ils sont limités à des zones ayant de petits ratios de formes. En effet, ils détectent mal les formes rectangulaires. Ainsi, en utilisant des segments et des liens, Seglink n'est pas limité en taille et peut détecter plusieurs bouts de mots, de mots ou de phrases qui seront liés ensemble par la suite.

Seglink s'appuie sur un CNN pré entraîné pour créer un réseau entièrement convolutif. Des prédicteurs sont ajoutés à 6 des couches pour détecter les segments et les liens à différentes échelles comme on peut l'observer à la figure 2.8.

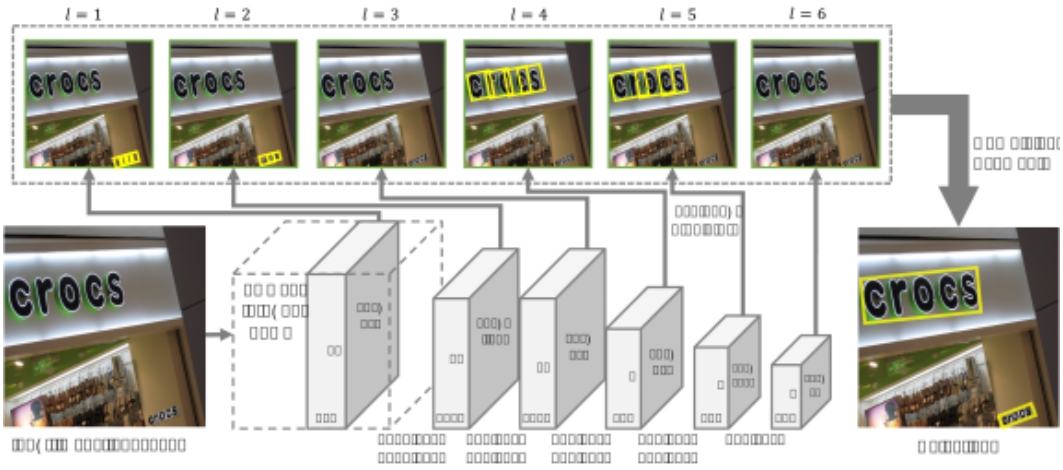

Figure 2.8: Detecting Oriented Text in Natural Images by Linking Segments. Extrait de Shi *et al.* (2017)

Finalement, les auteurs consolident les segments ensemble avec les liens en utilisant un algorithme de type *depth-first-search* (DFS) (Kozen, 1992) et les combinent en mots ou en phrases.

Seglink apporte une solution intéressante, car elle ne se base que sur un CNN modifié et elle permet de lier du texte qui sort des zones d'intérêts des SSD ou des CRNN. En effet, Seglink peut joindre des bouts de texte détectés et les assembler,

supportant ainsi du texte ayant des formes incurvées par exemple. L'approche est rapide et précise : sur le jeu de données ICDAR 2013 (Karatzas *et al.*, 2013a), elle obtient 85,3 % de F-Mesure à 20,6 images par seconde.

Cependant, Seglink échoue lui aussi à détecter du texte ayant beaucoup d'espacement entre les lettres ou encore du texte trop incurvé ou vertical.

Rosetta : Large scale system for text detection and recognition in images (Borisyuk et al.*, 2018)*
Rosetta se base sur *Faster R-CNN* pour l'extraction de texte et sur un *Convolutional Recurrent Neural Network (CRNN)* pour la reconnaissance de texte. Faster R-CNN utilise un CNN pour extraire des cartes d'attributs, un *Region proposal network (rpn)* qui produit des boîtes de coordonnées qui ont possiblement du texte, et finalement, extrait les caractéristiques des cartes d'attributs par rapport aux coordonnées proposées et les classifie. Une fois la classification terminée, l'algorithme de NMS élague les boîtes de texte pour extraire celles qui ont le plus haut taux de confiance.

Rosetta semble être très prometteur puisqu'il se base sur des méthodes éprouvées de détection d'objets. Toutefois, il considère que toutes les images, en entrée, ont la même taille. Pour de longs mots, ceci pose problème, puisque le texte sera redimensionné pour qu'il rentre dans le format d'entrée. Inversement, on utilisera du «padding» pour insérer le petit texte.

Finalement, les jeux de données permettant d'atteindre les prétendus résultats ne sont pas rendus publics, il n'y a pas de code disponible publiquement et les auteurs ne se comparent pas aux solutions récentes.

TextBoxes++ : A Single-Shot Oriented Scene Text Detector (Liao et al.*, 2018)*
Comme son titre le laisse présager (++) est une nouvelle itération de Liao *et al.*

(2017). Les auteurs proposent 4 nouvelles contributions : i) un système de détection multi orienté, encore basé sur SSD, ii) l'optimisation de l'architecture et de certains algorithmes (NMS, couches textboxes Liao *et al.* (2017)), iii) la comparaison exhaustive avec les solutions les plus récentes, iv) une nouvelle fonction de score qui combine la détection et la reconnaissance.

Liao *et al.* (2018) proposent l'utilisation des «quadrilateral default boxes», qui sont des «defaults boxes» ayant subi des rotations. Un «default box» est une région dans l'image. Même s'il s'agit d'un pas dans la bonne direction pour la détection de texte orienté, l'algorithme manque de données pour l'entraîner et il ne détecte pas le texte avec des orientations en demi-cercle, ou avec des formes plus exotiques.

Les auteurs proposent d'utiliser le score de la reconnaissance de texte pour améliorer les résultats de la détection lors de l'entraînement, car cela aide à enlever les faux positifs dans la détection.

Real-time Scene Text Detection with Differentiable Binarization (Liao et al.*, 2019)* Cette étude, selon les mêmes auteurs que Liao *et al.* (2018), tente une nouvelle approche. Ils s'inspirent de la nouvelle tendance d'utiliser des méthodes de segmentation d'image dans la détection de texte. La segmentation permet de mieux supporter le texte de différente taille, forme et orientation. Ainsi, ils proposent un module s'appelant *Differentiable Binarization (DB)* qui transforme une image en noir et blanc, selon un seuil. Ce module est intéressant, car il rend la fonction de *binarisation* dérivable, donc entraînable avec un réseau de segmentation et la règle de dérivation en chaîne. Ceci n'est pas sans rappeler l'algorithme MSER qui détecte les formes selon des seuils de gris.

Ainsi, Liao *et al.* (2019) apportent une contribution significative grâce à une équation de binarisation différentiable et une matrice de seuils adaptatifs où chaque

pixel a un différent seuil de binarisation. Ceci leur permet d'obtenir une F-mesure de 82,8%, et de traiter 62 images par secondes, sur le jeu de données Total-Text (Ch'ng *et al.*, 2020).

Toutefois, les auteurs ont identifié une faiblesse de leur approche : BD ne peut détecter le texte dans le texte, c'est à dire, si du texte se retrouve incrusté à l'intérieur d'un autre texte.

TextFuseNet : Scene Text Detection with Richer Fused Features (Ye et al.*, 2020)* Dans cet article, nous retournons aux détecteurs à plusieurs étapes, car les auteurs utilisent Detectron2 (Wu *et al.*, 2019) un framework développé par Facebook et qui lui-même tire ses origines de Faster R-CNN and Mask R-CNN Massa et Girshick (2018).

Ainsi, les auteurs innovent en utilisant une solution qui ne prend pas qu'une seule région d'intérêt en considération : l'idée de garder un contexte global à l'image permet de réduire les faux positifs.

Pour ce faire, ils proposent d'utiliser une nouvelle représentation, soit une fusion des caractères, des mots et du contexte sémantique général. Cette représentation à plusieurs niveaux peut mieux décrire le texte dans une image et est plus robuste.

Comme on peut le voir sur la figure 2.9, TextFuseNet utilise 3 niveaux de représentation. Ensuite, TextFuseNet collecte et consolide les attributs de texte des différents niveaux en tirant parti du *multi-path fusion architecture*, qui permet d'aligner et de fusionner différentes représentations. Toutefois, comme les auteurs le mentionnent, ce n'est pas tous les jeux de données qui sont annotés au niveau des caractères dans les images. Pour faire face à ce problème, ils utilisent l'apprentissage semi-supervisé (voir Chap. 1) pour annoter les données manquantes.

Ceci permet à TextFuseNet d'apprendre de plusieurs types et formes de texte et

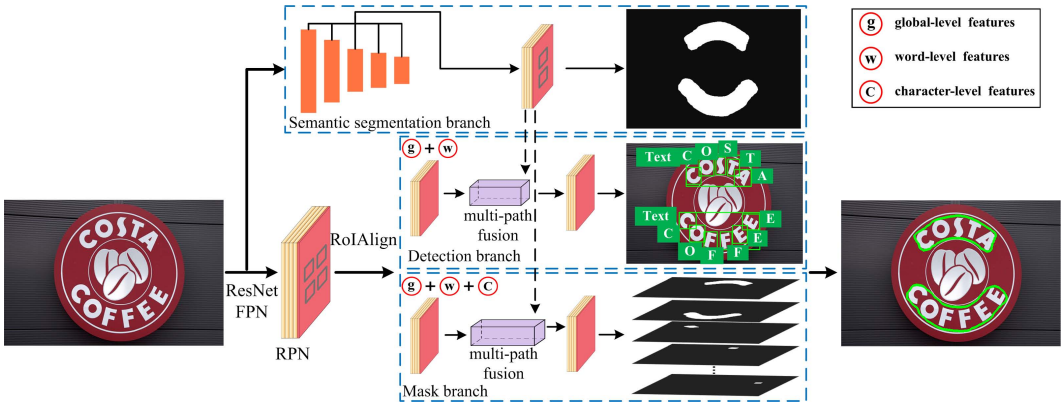

Figure 2.9: TextFuseNet : Scene Text Detection with Richer Fused Features. Extrait de Ye *et al.* (2020)

ainsi d'obtenir de meilleurs résultats dans la détection de texte. De fait, les auteurs obtiennent une F-Mesure de 94,3% sur le jeu de données ICDAR 2013 (Karatzas *et al.*, 2013a), 92,1% sur ICDAR 2015 (ICD, 2015), 87,1% sur Total-Text (Ch'ng *et al.*, 2020).

Un récapitulatif (Khan *et al.*, 2021) des derniers modèles de détection est présenté ci-après.

2.4 Reconnaissance optique de caractères

Ici, nous nous intéressons à la reconnaissance optique de caractère car on tente d'identifier le texte que nous avons détecté dans la section précédente.

Classification temporelle connexionniste (CTC) (Graves et al.*, 2007)*

L'algorithme de la CTC est basé sur une procédure *avant-arrière* qui permet de garder le contexte des entrées. On l'utilise pour entraîner des modèles récurrents à étiqueter des séquences sans segmentation précise. Ainsi, CTC cherche à prédire la meilleure étiquette pour tous les chemins possibles, on peut le voir comme une

Method	Year	ICDAR'11				ICDAR'13				ICDAR'15			
		P	R	F-M	T(s)	P	R	F-M	T(s)	P	R	F-M	T(s)
Lu et al. (2015)*	2015	–	–	–		0.89	0.69	0.78		–	–	–	–
CTPN (Tian et al. 2016a)	2016	0.89	0.79	0.84		0.93	0.83	0.88	0.14 s	0.74	0.52	0.61	–
Gupta et al. (2016)	2016	0.92	0.75	0.82	0.25 s	0.92	0.76	0.83	0.25 s	–	–	–	–
He et al. (2016a)	2016	0.91	0.74	0.82	–	0.93	0.73	0.82	–	–	–	–	–
Qin and Manduchi (2017)	2017	–	–	–		0.90	0.83	0.86	0.45 s	0.79	0.65	0.71	0.45 s
SegLink (Shi et al. 2017a)	2017	–	–	–	–	0.87	0.83	0.85	0.05 s	0.73	0.77	0.75	–
TextBoxes (Liao et al. 2017)	2017	0.89	0.82	0.86	0.73 s	0.89	0.83	0.86	0.73 s	–	–	–	
R^2CNN (Jiang et al. 2017)	2017	–	–	–	–	0.93	0.82	0.87	–	0.85	0.79	0.82	2.25 s
Neycharan and Ahmadyfard (2018)*	2017	–	–	–	–	0.93	0.77	0.85	–	–	–	–	–
Dey et al. (2017)*	2017	–	–	–	–	0.67	0.87	0.75	–	–	–	–	–
He et al. (2017c)	2017	–	–	–	–	0.92	0.81	0.86	0.9 s	0.82	0.80	0.81	–
EAST (Zhou et al. 2017)	2017	–	–	–	–	–	–	–	–	0.83	0.78	0.81	0.06 s
Li et al. (2017)+	2017	0.89	0.81	0.85	0.73 s	–	–	–	–	0.91	0.80	0.85	0.73 s
Lyu et al. (2018a)	2018	–	–	–	–	0.92	0.84	0.88	1 s	0.89	0.79	0.84	1 s
RRPN (Ma et al. 2018)	2018	–	–	–	–	0.95	0.88	0.91	–	0.84	0.77	0.80	–
FOTS (Liu et al. 2018c)+	2018	–	–	–	–	–	–	0.92	0.04 s	0.91	0.85	0.88	0.12 s
Textboxes ++ (Liao et al. 2018a)	2018	–	–	–	–	0.92	0.86	0.89	–	0.87	0.78	0.83	0.43 s
PSENet (Li et al. 2018)	2018	–	–	–	–	–	–	–	–	0.89	0.85	0.87	0.13 s
Tang and Wu (2018)	2018	0.90	0.84	0.87	8.7 s	0.91	0.86	0.88	–	–	–	–	–
Sain et al. (2018)*	2018	–	–	–		0.87	0.74	0.80	–	–	–	–	–
MCN (Liu et al. 2018a)	2018	–	–	–	–	0.88	0.87	0.88	0.03 s	0.72	0.80	0.76	–
PixelLink (Deng et al. 2018)	2018	–	–	–	–	0.88	0.87	0.88	–	0.85	0.82	0.84	0.33 s
Richardson et al. (2019)	2019	–	–	–	–	–	–	–	–	0.85	0.83	0.84	0.28 s
RCR-CNN (Zhu et al. 2019)	2019	–	–	–	–	–	–	–	–	0.88	0.86	0.87	0.55 s
SegLink++ (Tang et al. 2019)	2019	–	–	–	–	–	–	–	–	0.83	0.80	0.82	0.14 s

Figure 2.10: Comparaison des différents modèles de détection réalisés par Khan *et al.* (2021)

façon de faire le ménage dans les étiquettes en supprimant les répétitions et ensuite les blancs (—) comme l'illustre l'exemple qui suit :

$$b\ b\ \text{——}\ e\ e\ \text{—}\ b\ e\ \longmapsto bebe \tag{2.5}$$

Le réseau est donc entraîné à maximiser la probabilité de la séquence d'étiquettes sachant la séquence d'entrée.

An End-to-End Trainable Neural Network for Image-based Sequence Recognition and Its Application to Scene Text Recognition (CRNN) (Shi *et al.*, 2015)
Shi *et al.* (2015) proposent l'utilisation d'un *Convolutional Recurrent Neural Network (CRNN)*, un réseau à 3 couches, soit la couche convolutionnelle, la couche récurrente et la couche de traduction.

Au début, la couche CNN extrait les cartes d'attributs des images d'entrée pour ensuite les passer à un réseau de neurones récurrents (RNN). Le RNN prédit le texte pour chacune d'entre elles. Et finalement, la couche de transcription traduit les prédictions séquentielles du RNN en texte.

Même si le CRNN est composé de plusieurs types de réseaux, il peut être entraîné de bout en bout avec une seule fonction de score : la fonction de coût *classification temporelle connexionniste (CTC)* propage les erreurs de bout en bout (Graves *et al.*, 2007).

Un classifieur RNN est entraîné à prédire la probabilité de la position à l'instant t d'un caractère. Ceci pose problème si les séquences ne sont pas parfaitement segmentées. Par exemple, dans l'écriture manuscrite une personne pourrait avoir tendance à écrire, par habitude ou coquetterie, la lettre « z » plus grosse que les autres lettres.

L'algorithme CTC vient ainsi résoudre le problème quand on ignore l'alignement au préalable. Il ne nécessite pas de segmentation initiale, car il extrait la séquence la plus probable de caractères en sortie.

Towards Accurate Scene Text Recognition with Semantic Reasoning Networks (SRN) (Yu *et al.*, 2020)

Dans cet article, Yu *et al.* (2020) mentionnent qu'il y a peu d'évolution dans les modèles, entre autres parce qu'ils fonctionnent déjà assez bien. Ils mentionnent cependant que ces modèles basés sur les RNN ont des lacunes telles que la perte de contexte sémantique par exemple. Ils proposent donc un réseau de raisonnement sémantique (SRN) qui peut garder le contexte sémantique global grâce à leur module (GSRM). Le résultat est intéressant, car ils obtiennent de meilleurs résultats que les autres modèles sur 7 jeux de données.

Ce système est très complexe et diffère beaucoup des méthodes utilisées dans cette recherche. Il est cependant incontournable car il obtient les meilleurs résultats sur les jeux de données ICDAR 2013 (Karatzas *et al.*, 2013a) et SVT (Babenko *et al.*, 2011) en 2020 comme nous pouvons le voir dans le tableau 2.11.

	Method	ConvNet,Data	Annos	IC13	IC15	IIIT5K	SVT	SVTP	CUTE
Semantic context -free	Jaderberg *et al.* [14]	VGG,90K	word	90.8	-	-	80.7	-	-
	Jaderberg *et al.* [12]	VGG,90K	word	81.8	-	-	71.7	-	-
	Shi *et al.* [29] (CTC)	VGG,90K	word	89.6	-	81.2	82.7	-	-
	Lyu *et al.* [25] (Parallel)	ResNet,90K+ST	word	92.7	76.3	94.0	90.1	82.3	86.8
	Xie *et al.* [39] (ACE)	VGG,90K	word	89.7	68.9	82.3	82.6	70.1	82.6
	Liao *et al.* [22] (FCN)	ResNet,ST	word,char	91.5	-	91.9	86.4	-	-
Semantic context -aware	Lee *et al.* [19]	VGG,90K	word	90.0	-	78.4	80.7	-	-
	Cheng *et al.* [5] (FAN)	ResNet,90k+ST	word	93.3	70.6	87.4	85.9	-	-
	Cheng *et al.* [6] (AON)	self,90k+ST	word	-	68.2	87.0	82.8	73.0	76.8
	Bai *et al.* [3]	ResNet,90K+ST	word	94.4	73.9	88.3	87.5	-	-
	Yang *et al.* [41]	VGG,90K+self	word,char	-	-	-	-	75.8	69.3
	Shi *et al.* [31] (ASTER)	ResNet,90K+ST	word	91.8	76.1	93.4	89.5	78.5	79.5
	Zhan *et al.* [44] (ESIR)	ResNet,90K+ST	word	91.3	76.9	93.3	90.2	79.6	83.3
	Yang *et al.* [40] (ScRN)	ResNet,90K+ST	word,char	93.9	78.7	94.4	88.9	80.8	87.5
	Li *et al.* [20] (SAR)	ResNet,90K+ST	word	91.0	69.2	91.5	84.5	76.4	83.3
	Liao *et al.* [21] (SAM)	ResNet,90K+ST	word	95.3	77.3	93.9	90.6	82.2	**87.8**
Ours	SRN w/o GSRM	ResNet,90K+ST	word	93.2	77.5	92.3	88.1	79.4	84.7
	SRN	ResNet,90K+ST	word	**95.5**	**82.7**	**94.8**	**91.5**	**85.1**	**87.8**

Figure 2.11: Comparaison des différents modèles de reconnaissance réalisée par Yu *et al.* (2020)

2.5 Post traitement de la reconnaissance

La détection et la reconnaissance de texte dans les images naturelles sont, comme nous l'avons remarqué jusqu'à présent, loin d'être parfaites. Les résultats sont souvent truffés d'erreurs ou partiels.

Pour résoudre en partie ce problème, il est possible d'utiliser le traitement automatique du langage (TALN). Les technologies associées à ce sujet ont aussi largement profité des avancées de l'intelligence artificielle.

Dans ce chapitre, nous faisons donc un tour d'horizon de l'état de l'art du TALN relié à notre sujet de recherche, en présentant les travaux qui s'intéressent aux modèles de langue.

*Modèles de langage neuronaux (**Neural Embeddings**)*

Les plongements de mots ont été introduits grâce à Bengio *et al.* (2003) dans leurs recherches sur les modèles de langage neuronaux. Ils ont baptisé ce processus *learning a distributed representation for words*. Le réseau doit apprendre à estimer la probabilité du prochain mot. Ce système est réalisé en deux temps, premièrement, il doit associer chaque mot du vocabulaire à un vecteur distribué de caractéristiques, c'est-à-dire un vecteur de nombres réels qui est le paramètre d'entrée de la deuxième couche cachée du réseau. Ensuite, en sortie, la probabilité n-gramme de chaque mot dans une liste de mots est calculée pour prédire le mot suivant (figure 2.12). Donc l'idée est que les réseaux de neurones apprennent les caractéristiques des mots qu'on extrait pour en faire des plongements de mots (ou *word embeddings* en anglais).

Word2vec (Mikolov *et al.*, 2013)

Vers 2013, Mikolov *et al.* (2013) proposent CBOW et skip-gram. CBOW est une approche neuronale pour construire des plongements de mots et l'objectif est de calculer la probabilité d'un mot étant donné les mots du contexte. Inversement, skip-gram doit prédire les mots du contexte sachant un mot *centre*. Pour les 2 modèles, les *word embeddings* sont déterminés à la couche cachée du réseau.

Une limitation de cette solution est que chaque mot est un vecteur indépendant. On ne peut pas combiner un ensemble de mots comme "lait au chocolat" ou "Journal de Montréal". Il ne reconnaît pas les variations dans les mots, ainsi "Attends" et "Attendons" sont considérés comme étant totalement différents. De plus, Word2vec n'a aucune idée de comment interpréter de nouveaux mots, il les transforme en vecteur aléatoire, ce qui est loin d'être idéal.

Global Vectors for Word Representation (GloVe) (Pennington *et al.*, 2014)

Global Vectors for Word Representation (GloVe) est une approche introduite

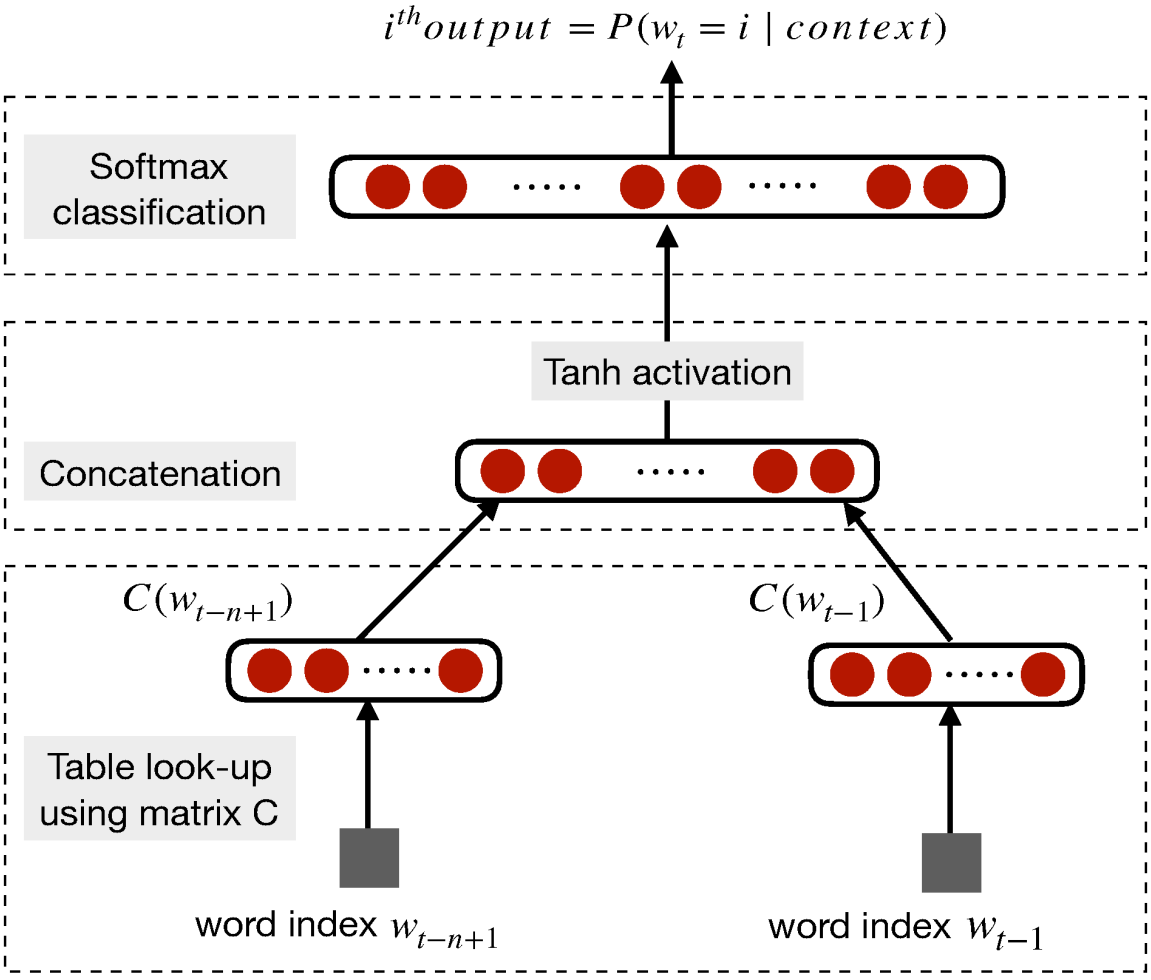

Figure 2.12: Modèle de langage probabiliste neuronal (Bengio *et al.*, 2003)

par Pennington *et al.* (2014). Glove applique une probabilité *word-word co-occurence* pour bâtir les plongements de mots. Autrement dit, si 2 mots coexistent souvent, il sont probablement liés d'une certaine manière.

Contrairement à Word2vec, GloVe prend en compte toute l'information d'un corpus donné et non pas seulement un petit ensemble de mots mais le contexte dans lequel apparaît le mot.

FastText (Bojanowski *et al.*, 2017)

Bojanowski *et al.* (2017) de Facebook propose d'améliorer Word2vec en apprenant la représentation vectorielle de chaque mot et les n-grammes de ceux-ci. Les valeurs sont ensuite regroupées dans un seul vecteur. Malgré le coût et la complexité que ça ajoute à l'entraînement, cela enrichit les *word embeddings* en y ajoutant de l'information sur les sous-mots. Les vecteurs **FastText** ont démontré qu'ils sont plus précis que **Word2vec** hormis sémantiquement en anglais ou Word2vec obtient de meilleurs résultats (Bojanowski *et al.*, 2017). Un autre avantage de FastText est qu'il permet de représenter des mots inconnus et des parties de mots.

Contextual word embeddings

Jusqu'à présent, aucune des méthodes ne prend en compte efficacement le contexte des mots. Par exemple, chaque mot ne peut avoir qu'une seule représentation vectorielle. Ceci pose problème puisqu'un mot aura toujours la même représentation, même selon le contexte comme "New York Times". Finalement, même pour les mots qui n'ont qu'un sens, dépendamment du contexte, de la culture et autre, leurs sens peuvent changer.

ELMo : Embeddings from Language Models (Peters et al.*, 2018)*

Peters *et al.* (2018) proposent un nouveau type de *contextual word embeddings* qui permet de modéliser des caractéristiques plus complexes des mots (syntaxique et sémantique) et selon différents contextes linguistiques. Pour ce faire, ils utilisent un *bidirectional langage model (biLM)* : un modèle de langage récurrent (Hochreiter et Schmidhuber, 1997) pré entraîné de quelques couches et bidirectionnel qui extrait les poids à chaque couche selon les mots en entrées. Ensuite, un poids normalisé est calculé pour obtenir les plongements de mots.

Il permet des améliorations sur 6 tâches de TALN, allant de l'analyse de sentiment à la reconnaissance d'entités nommées selon Peters *et al.* (2018).

Finalement, les auteurs font aussi mention que chaque couche du **BiLM** extrait des caractéristiques différentes du texte. Par exemple, la première est meilleure pour le *part-of-speech tagging* et la deuxième pour la désambiguïsation.

BERT : Bidirectional Encoder Representations from Transformers (Devlin et al.*, 2018)*
Un an plus tard, BERT est sortie et permet une amélioration sur 11 tâches, dont notamment la classification de texte et les question réponses (Devlin *et al.*, 2018). BERT est conçu en 2 étapes : le pré entraînement et l'ajustement. Dans le pré entraînement, BERT est entraîné sur des données non annotées sur un *masked langage model (MLM)* : on choisit un mot au hasard qu'on cache et on essaie de le prédire. Dans la phase de *fine-tuning*, BERT part avec les paramètres pré entraînés et les ajuste sur les autres tâches (Reconnaissance d'entités nommées (NER), marquage d'une partie du discours (POS), etc.).

Bien que conceptuellement BERT soit similaire à ELMo, il se différencie de plusieurs façons : il est capable de modéliser les relations entre 2 phrases, il utilise une architecture basée sur les *transformers* et finalement il utilise un *masked langage model*. En cachant des mots, il permet au modèle d'être plus robuste.

Pour conclure, voici quelques caractéristiques de BERT :

— BERT est gigantesque, il a 340 millions de paramètres, 1024 couches, etc.
— Il est entraîné sur un corpus de 3.3 milliards de mots.
— Il est entraîné sur 16 TPU

Simple Applications of BERT for Ad Hoc Document Retrieval (Yang et al.*, 2019)*
Étant donné les résultats impressionnants de BERT dans le monde du classement de passages (*Passage ranking*) et surtout du Q&A, Yang *et al.* (2019) explorent comment utiliser BERT pour améliorer la recherche de document. Le classement

de documents est la tâche dans laquelle des documents ou des passages de texte sont récupérés et classés à partir d'une collection de documents en fonction de leur pertinence par rapport au besoin d'information d'une requête ou d'une question spécifique formulée par les utilisateurs. Ainsi, grâce au classement de passage, la pertinence d'un document peut être approximée par le meilleur passage dans un document, et donc permet d'obtenir de meilleurs résultats de recherche quand on utilise un modèle de langage plus robuste. Par exemple, l'algorithme sera capable de déterminer si dans un document qui parle de légume si un passage est spécifiquement dédié aux tomates.

Donc l'idée principale de cet article est de combiner le score de la recherche de document avec le score de passage de BERT.

2.6 Correction automatique de la reconnaissance

Parallel Iterative Edit Models for Local Sequence Transduction (Awasthi et al.*, 2019)*
Awasthi *et al.* (2019) présentent leur solution qui utilise l'architecture BERT (Devlin *et al.*, 2018) pour faire de la correction grammaticale (GEC). Toutefois, contrairement aux solutions communes de GEC qui classifient en le texte ayant des erreurs ou non, PIE tente de proposer des corrections à ce texte.

Ainsi, suite aux modifications de BERT, PIE produit des *edits* (copier, effacer, ajouter, remplacer, etc.) à la place de *tokens* (des mots de remplacements). Par exemple, pour une phrase telle que *"stap tte cor"* les modèles standard de séquences à séquences proposeraient des *tokens* "*stop the car*". PIE, quant à lui, propose des modifications telles par exemple : *mettre en majuscule la première lettre du premier mot remplacer (a) par (o) du premier mot remplacer (tt) par (th) du deuxième mot remplacer (o) par (a) du troisième mot*

Ensuite, à la place de produire une séquence de texte généré avec les *tokens* proposés, PIE conserve la séquence annotée et la corrige comme nous l'avons vu dans l'exemple ci-haut. Ensuite, il repasse la séquence, pour raffiner la correction et finalement, Awasthi *et al.* (2019) adaptent BERT en transformant la couche logique avec les *edits* et les corrections.

Les auteurs mentionnent qu'ils ont testé PIE sur un jeu de données en suédois et obtiennent des résultats similaires aux solutions auxquelles ils se comparent, mais sans plus. Il semble par contre que PIE soit significativement plus rapide, obtenant un gain de 20 mots/s.

From the Paft to the Fiiture : a Fully Automatic NMT and Word Embeddings Method for OCR Post-Correction (Hämäläinen et Hengchen, 2019)
Dans cet article, Hämäläinen et Hengchen (2019) présentent une méthode d'apprentissage automatique sans supervision qui se base sur les avancées en *neural machine translation (NMT)*. Ils proposent une solution fondée sur les plongements de mots, une liste de lemmes (Un lemme fait référence à un mot, de quelque nature que ce soit) et un lemmatiseur qui extrait des données parallèles pour entraîner le modèle NMT.

Les modèles NMT nécessitent des données annotées avec les erreurs et leurs corrections, donc la première étape consiste à extraire les similarités sémantiques en entraînant un modèle Word2Vec Mikolov *et al.* (2013). Toutefois, au lieu de trouver le bon mot pour un mot en erreur, il fait l'inverse : friendship obtient friendlhip, friendihip, friendflip, friend-, affection, friendthip, gratitude, affetion, friendflhip et friendfiip.

Toutefois, cette façon de faire peut entraîner des erreurs, car des petits mots ou des mots qui se ressemblent peuvent être corrigés par erreur. Ainsi, les auteurs utilisent un jeu de données pour l'entraîner sur de vraies erreurs d'OCR, ce qui

permet de réduire les mauvaises données. De plus, ils donnent plus de poids aux longs mots pour lesquels on peut être plus sûr de la bonne correction car ils présentent moins d'ambiguïté.

Finalement, Hämäläinen et Hengchen (2019) utilisent un modèle NMT pour transformer le texte au niveau des caractères. Une fois faits, ils vérifient que les mots proposés sont des mots qui existent dans *Oxford English Dictionary* et prennent celui ayant le plus probable. Les auteurs mentionnent que leur solution ne peut toutefois pas corriger les mots collés ensemble.

CHAPITRE III

MÉTHODOLOGIE ET DONNÉES

3.1 Approche

Le but principal de cette recherche est d'extraire des informations textuelles de qualité de la collection de films de l'Office national du film du Canada (ONF) pour ensuite les indexer et les rendre disponibles dans un moteur de recherche capable d'effecteur des requêtes complexes.

Nous pouvons diviser notre problématique en plusieurs modules et en faire un flux de tâches : l'ingestion des images, la détection de texte, la reconnaissance de texte, la post-correction du texte ainsi que la consolidation temporelle du texte, l'indexation du texte et finalement la recherche d'information dans un moteur de recherche. Chacun de ces modules est un sujet de recherche en soi.

Ce projet de recherche est donc multidisciplinaire, au contenu très riche en histoire, car nous traitons des vieux films de l'ONF. Nous tentons d'optimiser nos solutions pour chacun de ces films comme étant des cas particuliers et parfois uniques. Chaque étape du flux de travail qui permet la recherche de texte dans les films est une discipline complexe et non linéaire.

3.2 Contribution

Il semble que tous les gagnants des compétitions en détection d'objets et de texte utilisent d'une façon ou d'une autre l'apprentissage profond. Notre sujet s'intéresse à la détection et à la reconnaissance de texte dans les films. Nous proposons l'utilisation et l'adaptation de modèles de l'état de l'art pour la détection et la reconnaissance de texte dans les films. Pour ce faire, nous proposons des nouveaux jeux de données, mais aussi un module de post-correction et finalement un intégration des différente tâches.

Un des intérêts de cette recherche est de reconnaître le texte incrusté dans l'image des vieux films ou des vidéos, par exemple, les sous-titres. Nous proposons donc un système qui émule l'incrustation de sous-titres dans les films et par le fait même un jeu de données de 100 films.

De plus, pour s'assurer de la validité des expérimentations, nous créons un deuxième jeux de données (ONF-TEXT) annotés à la mains qui permet de tester les cas les plus difficiles que nous avons à l'ONF.

Au cours de nos expérimentations, nous avons découvert que les systèmes de reconnaissance de texte avaient tendance à faire les mêmes erreurs de reconnaissance. Par exemple, en anglais «tbe» à la place de «the». Pour résoudre ce problème, nous proposons un système de post traitement (postocr) de la reconnaissance qui corrige les erreurs de reconnaissance par des heuristiques. De plus, postocr rassemble le texte similaire et proche temporellement pour recréer des fichiers de sous-titres et pour extraire le texte le plus probable pour un laps de temps.

3.3 Suivi d'expériences et reproductibilité

En science, il est primordial de pouvoir évaluer rigoureusement les approches pour démontrer la progression ou encore l'échec des expérimentations.

Toutefois, comme le fait remarquer Yin *et al.* (2016), les erreurs dans les jeux de données, les biais, les environnements de test, les hyperparamètres, pour ne nommer que ceux-là, rendent difficile la comparaison des différentes solutions.

Selon Ian Hogarth (2020) seulement 15% des études publient leur code. Ce pourcentage est encore pire lorsqu'on se penche sur les études publiées par l'industrie. Nous avons d'ailleurs pu le constater plusieurs fois en réalisant l'état de l'art (par exemple Facebook Rosetta).

Même quand le code est publié, il est parfois impossible de reproduire les travaux, soit parce que les données ne sont pas disponibles ou encore que le modèle pré entraîné n'est pas en ligne.

Un autre problème est la gestion des environnements. Quelle version de Pytorch (Wallach *et al.*, 2019) fut utilisée ? Est-elle compatible avec les dernières rustines Nvidia ? Ça devient rapidement un casse-tête.

Pour atténuer ces difficultés, on peut se tourner du côté de l'ingénierie logicielle qui fait face à ces défis depuis longtemps. En utilisant Docker (un logiciel libre permettant de lancer des applications dans des conteneurs logiciels (Merkel, 2014)), pour reproduire l'environnement de test et en utilisant des tests unitaires, nous pouvons reproduire les résultats plus facilement.

De plus, pour collaborer et faire un suivi de la progression de la solution, nous utilisons un logiciel de gestion de versions comme Git par exemple.

Finalement, la documentation devrait fournir les dépendances et les étapes pour reproduire le projet. Il existe plusieurs autres meilleures pratiques dans les projets d'intelligence artificielle (Aaron Courville, 2019) et plusieurs chercheurs s'y penchent.

3.4 Les données

3.4.1 Les données à l'Office national du film du Canada

L'ONF dispose d'une collection active d'environ 14 000 oeuvres cinématographique : 10 000 pellicules (film chimique, par exemple le format 35mm), 3000 vidéos (bande magnétique, par exemple le format Betacam) et environ 1000 oeuvres entièrement numériques.

L'ONF a aussi 4000 heures réparties sur 60 000 plans d'archives et plus de 300 000 plans qui n'ont pas encore été indexés dans la collection. Ces plans reposent dans les voûtes en attendant d'être utilisés.

Pour ces ensembles, l'ONF ne dispose pas de jeux de données annotées et il est important de créer un échantillonnage représentatif des données et d'y inclure nos cas difficiles pour être certain de bien effectuer nos expérimentations.

Pour ce faire, deux grandes tendances dans le monde de la détection et de la reconnaissance de texte nous ont influencés : la génération de données synthétiques et l'apprentissage semi-supervisé (voir Chap. 1). Ces données sont ainsi devenues la pierre d'assise de ce mémoire et tous les jeux de données que nous décrivons dans cette section furent utilisés lors de nos tests et lors de l'entraînement des modèles.

3.5 Jeux de données

Dans cette partie, nous avons fait un recensement de tous les jeux de données utiles à notre travail. Il en existe beaucoup d'autres, mais nous n'allons prendre que l'alphabet latin, car nous n'avons pas ou très peu de données dans d'autres alphabets.

Tous ces jeux de données sont libres de droits et disponibles en ligne, sauf ceux de l'ONF. De plus, étant donné la rareté des données, ils sont tous utilisés dans les articles couverts dans l'état de l'art et dans nos expérimentations.

Tableau 3.1: Comparaison des jeux de données utilisés dans le cadre de ce mémoire.

Jeux de données	Année	# Detection			# OCR		Orientation			Caractéristiques		Task	
		Entraînement	Test	Total	Entraînement	Test	H	MO	Cu	Langue	Étiquette	D	R
IC03* (Lucas *et al.*, 2003)	2003	258	251	509	1156	1110	✓	–	–	EN	W,C	✓	✓
SVT* (Babenko *et al.*, 2011)	2010	100	250	350	–	647	✓	–	–	EN	W	✓	✓
IC11 (El Abed *et al.*, 2011)	2011	100	250	350	211	514	✓	–	–	EN	W,C	✓	✓
IIIT 5K-words* (Jawahar *et al.*, 2012)	2012	–	–	–	2000	3000	✓	–	–	EN	W	–	✓
SVT-P* (Phan *et al.*, 2013)	2013	–	238	238	–	639	✓	✓	–	EN	W	✓	✓
ICDAR13* (Karatzas *et al.*, 2013a)	2013	229	233	462	848	1095	✓	–	–	EN	W	✓	✓
CUT80* (Wang et Lian, 2020)	2014	–	80	80	–	280	✓	✓	✓	EN	W	✓	✓
COCO-Text* (Veit *et al.*, 2016)	2014	43686	20000	63686	118309	27550	✓	✓	✓	EN	W	✓	✓
ICDAR15* ICD (2015)	2015	1000	500	1500	4468	2077	✓	✓	–	EN	W	✓	✓
ICDAR17 (Iwamura *et al.*, 2017)	2017	7200	9000	18000	68613	–	✓	✓	✓	ML	W	✓	✓
TotalText* (Ch'ng *et al.*, 2020)	2017	1255	300	1555	–	11459	✓	✓	✓	EN	W	✓	✓
SynthText* (Gupta *et al.*, 2016)	2016	800k	–	800k	8M	–	✓	✓	✓	EN	W	✓	✓
MJSynth Jaderberg *et al.* (2014b)	2014	–	–	–	8.9M	–	✓	✓	✓	EN	W	–	✓
ONF-Text*	2021	–	–	50	–	50	✓	✓	✓	ML	W	✓	✓

Note : * Ces jeux données furent utilisés lors des expérimentations. H : Horizontale, MO : Multi-Orienté, Cu : Incurvé, EN : Anglais, ML :Multilingue, M : Mort, C : Caractère, TL : Textline D : Detection, R : Reconnaissance.

3.5.1 Jeux de données créé dans le cadre du mémoire

ONF-SynthText

En s'inspirant de Gupta *et al.* (2016), qui génère des données synthétiques, nous avons décidé de créer un jeu de données qui émule le texte incrusté dans l'image des vieux films. Sachant que les CNN contiennent des millions de paramètres et nécessitent énormément de données annotées pour être entraînées avec succès, l'idée de générer automatiquement énormément de données va de soi.

Nous avons donc eu l'idée de prendre des films de l'ONF et d'émuler la création de sous-titres brûlés dans ceux-ci. Il nous fallait extraire les images de films, extraire le texte des sous-titres, et tenter de générer des données selon nos différents cas de figure :

— Films : grain dans l'image, mauvaise opacité, mauvais contrastes
— Vidéos : contraste clair, contour noir et caractère blanc

Pour ce faire, nous avons créé un engin qui génère des données le plus réalistes possible (figure 3.2). Nous utilisons donc un ensemble de filtres et d'algorithmes de OpenCV (OpenCV, 2015) pour le réaliser. Il est bâti pour être modulaire. Ainsi, plus nous avancerons dans le projet, plus nous pourrons ajouter de nouveaux cas spéciaux.

Les données sont annotées au niveau des lettres, des mots et des blocs de texte comme on peut le voir à la figure 3.1. Nous utilisons plusieurs formats de coordonnées, car les algorithmes de détection sur lesquels nous avons expérimenté ne travaillent pas tous au niveau des lettres.

Figure 3.1: Exemple d'annotation de données : En vert, les phrases, en noir les mots et en rouge les lettres.

Nous utilisons aussi un système de tâches spécifié par Calcul Québec (*Slurm Workload Manager*), ceci nous permet d'utiliser leur super ordinateur pour générer les données. Ce fut important, car le traitement de chaque image peut prendre plusieurs secondes sur des ordinateurs modernes. Pour un film d'une heure, la génération pouvait prendre une semaine ! Tandis, qu'avec Calcul Québec, ce fut réglé en moins d'une semaine.

Nous avons fait une recherche dans la collection de l'ONF pour récupérer 100 oeuvres ayant des sous-titres et ayant été tournées en films. De plus, nous avons filtré ces oeuvres pour n'obtenir que les films qui n'avaient pas de sous-titrage incrusté. Il nous a été possible de générer 100 films de 30 minutes, en moyenne.

Malheureusement, comme les films ne sont pas libres de droits, il n'est pas actuellement possible de rendre public ce jeu de données. De plus, après plusieurs expériences, nous avons observé que le jeu de données comportait quelques faiblesses : occasionnellement on retrouvait du texte ailleurs que dans notre génération de données ce qui provoque un biais sur l'entraînement.

Figure 3.2: Exemple de données ONF-SynthText

ONF-Text

Prenant note des erreurs du passé, nous avons annoté un vrai jeu de données en prenant des cas réels. Dans le but d'obtenir une mesure qui sera pertinente pour évaluer les différents systèmes et stratégies, nous utilisons le plus d'exemples extrêmes possible (mauvaise opacité, texte incurvé, arrière-plan bruité, police, etc.). Ceci risque malheureusement de donner des chiffres médiocres, mais au moins nous aurons une mesure représentative. Nous avons donc annoté manuellement 35 images de films, de vidéos et d'animation qui représentent les données de l'Office national du film du Canada (figure 3.3).

Pour ce faire, nous avons utilisé LabelImg (Tzutalin, 2015) qui génère des données au format PASCAL VOC (Everingham *et al.*, 2009). Le format PASCAL VOC définit pour une image plusieurs boîtes de coordonnées et une étiquette associée (un mot dans notre cas). Cette annotation nous permet en premier lieu de générer un jeu de données pour la détection et un autre pour la reconnaissance comme nous allons le voir dans le prochain chapitre.

Figure 3.3: Exemple de données ONF-Text

CHAPITRE IV

EXPÉRIENCES ET ANALYSE DE RÉSULTATS

4.1 Modules proposés

Dans le but de trouver la meilleure solution et les meilleurs résultats, nous avons expérimenté plusieurs systèmes allant du simple OCR, à la vision par ordinateur (CV), à des méthodes hybrides alliant vision par ordinateur jusqu'à l'apprentissage automatique classique et finalement l'apprentissage profond.

Nous avons dû faire face a plusieurs difficultés et proposer des solutions. Certaines d'entre elles ne furent pas évaluées selon les mêmes critères que les solutions d'apprentissage profond. Elles furent parfois abandonnées et nous allons expliquer pourquoi. Nous allons aussi présenter ce qui a fonctionné et aborder la solution choisie et son intégration à l'ONF.

4.1.1 Extraction des images

Pour extraire les images, nous utilisons simplement le logiciel FFMpeg (Tomar, 2006). Lors de nos expérimentations, un des goulots d'étranglement était le traitement des films pour en extraire les images qui se faisaient en temps réel même si nous ne voulions en lire que quelques-unes par seconde OpenCV (2015).

Plus tard, nous avons découvert une façon d'accélérer considérablement le traite-

ment des fichiers vidéo MPEG-4 H264 : il est possible d'extraire des images aux temps τ seulement, en ignorant le reste grâce au *seek*. En effet, lisant seulement les keyframes, il peut accéder directement aux bonnes images.

4.1.2 Reconnaissance directement sur les images

La première expérience fut d'essayer des solutions OCR directement sur l'image ou encore en tentant de choisir des zones d'intérêts où le système pourrait obtenir un plus grand succès. Par exemple, en tentant de ne reconnaître que la zone des sous-titres.

Toutefois, cette solution pose problème lorsque le ratio d'image change. De plus les zones d'intérêts de texte sont dynamiques : pour rendre le texte plus visible, parfois les monteurs changent son positionnement.

De plus, la reconnaissance est défaillante, car les solutions commerciales ou libres de droits comme Tesseract (Kay, 2007) ne sont pas faites pour détecter et reconnaître du texte dans les images naturelles. Elles ont été entraînées pour reconnaître des caractères dans des documents texte noir et blanc.

Pour résoudre ce problème, nous avons tenté de traiter les images en utilisant des algorithmes comme la *binarization*, le *denoising*, le *blurring*, la *dilatation*, etc. Toutefois, les images ont trop de bruits (de défauts) et les résultats furent décevants. Un exemple de problème fut les arbres ou les branches, les algorithmes les détectent comme du texte et la reconnaissance ne produit que des erreurs.

En résumé, l'ensemble de ces étapes utilisant la reconnaissance directement sur les images étaient trop lentes pour le traitement de la collection. L'opération pouvait prendre parfois plus d'une minute pour traiter une image. Pour des milliers films d'environ 30 minutes, si l'on prend une image par 5 secondes, cela prendrait trop

de temps pour traiter l'ensemble de ces films.

Il devenait évident que notre principalement préoccupation devait se tourner vers la détection de texte dans les images. Cette première expérience était décevante, mais en parcourant l'état de l'art et grâce aux compétitions de l'ICDAR (ICD, 2015), plusieurs solutions nous étaient offertes.

4.2 Détection de texte

Dans le but de trouver le meilleur détecteur de texte, nous avons comparé différents modèles . Nous avons commencé par les plus simples, basés sur l'apprentissage automatique et la vision par ordinateur puis nous avons bifurqué vers l'apprentissage profond.

4.2.1 MSER : Maximally stable extremal regions

Dans cette itération, nous avons utilisé le MSER que nous avons présenté dans l'état de l'art. Toutefois, il est très lent, soit à 0,3 seconde par image ayant une résolution de 800x600 pixels. Cette méthode prendrait plus de 7 heures pour traiter un film d'une heure à 24 images par seconde. De plus, avec une F-Mesure 36% sur ICAR 2011 (El Abed *et al.*, 2011) pour la reconnaissance de bout en bout, les résultats ne sont pas assez fiables.

De manière qualitative, lors des expérimentations, nos observations concordent avec les conclusions de l'article. La détection fonctionne très bien, mais propose beaucoup de faux positifs et la reconnaissance fonctionne très mal, faisant face aux mêmes problématiques de l'OCR directement sur les images.

4.3 Apprentissage profond

La vraie expérimentation commence ici, car grâce à un jeu de données que nous avons annoté manuellement (ONF-Text) et une standardisation de notre méthodologie (calcul de la F-Mesure, rappel et précision) nous avons pu comparer les différents algorithmes de détection et de reconnaissance.

Dans nos expérimentations, nous avons utilisé plusieurs librairies d'apprentissage profond (Wallach *et al.*, 2019; Chollet *et al.*, 2015; Jia, 2013; Abadi *et al.*, 2015), pour réimplémenter les différents modèles que nous avons explorés dans l'état de l'art. Parfois, ces modèles étaient disponibles et libres de droits, ce qui allégeait notre travail et limitait l'introduction d'erreurs.

Toutefois, suite à l'étude exhaustive de l'état de l'art, nous avons pris des choix informés sur l'orientation que devraient prendre nos expérimentations. Ainsi, au moment de faire l'implémentation, la compétition ICDAR 2015 (Incidental Scene Text text detection) semblait avantager les solutions se basant sur les SSD et Textboxes venait (en 2017) d'obtenir le meilleur score (ICD, 2015; Liao *et al.*, 2017). Ce fut donc, notre point de départ. Il faut souligner que les détecteurs SSD (Liu *et al.*, 2015) sont plus polyvalents, présentant de bonnes performances par rapport à leur complexité.

Un tableau de toutes les approches sera fourni en fin de chapitre et nous y présenterons les résultats sur le jeu de données ONF-Text.

4.3.1 Approche basée sur TextBoxes (Liao *et al.*, 2017)

Dans la première itération de nos expérimentations, nous avons tenté de reproduire l'implémentation de TextBoxes de Liao *et al.* (2017). Pour ce faire, nous avons utilisé Keras (Chollet *et al.*, 2015) et Tensorflow (Abadi *et al.*, 2015). Nous avons

aussi voulu utiliser le code fourni avec l'article, mais il utilise Caffe (Jia, 2013) et le code n'est plus à jour, les pilotes graphiques ne sont plus compatibles.

Ainsi, nous entraînons TextBoxes avec des images de 300x300 pixels en utilisant la descente des gradients SGD avec un taux d'apprentissage de 10^{-3} qu'on réduit à 10^{-4} après 40k itérations et un momentum de 0.9.

TextBoxes est pré entraîné 50k fois sur SynthText (Jaderberg *et al.*, 2014b), ensuite 2000 fois sur ICDAR 2013 (Karatzas *et al.*, 2013a) et chaque image est augmentée. Toutefois, en réalisant l'expérimentation, nous avons remarqué que le jeu de données d'entraînement de l'ICDAR 2013 à seulement 229 images. Nous réalisons notre expérimentation sur un serveur Z820 ayant une carte GeForce GTX 1080 Ti.

Nous obtenons des résultats similaires à l'article sur les données ICDAR 2013, soit une F-mesure de 80,7% comparativement à 85% dans l'article. La différence provient peut-être de l'omission de données d'entraînement dans l'article ou encore des erreurs dans notre implémentation.

Toutefois, comme on peut le remarquer à la figure 4.1, TextBoxes n'arrive pas à détecter le texte incurvé ou encore le texte ou le contraste avec l'arrière-plan n'est pas clair. En conclusion,les résultats sur nos données sont nettement moins précis que dans l'article.

4.3.2 SegLink (Shi *et al.*, 2017)

Dans la deuxième itération de nos expérimentations, nous nous attaquons à SegLink (Shi *et al.*, 2017) en tentant de réimplémenter l'article. Selon l'article, Seglink est entraîné avec SynthText (Jaderberg *et al.*, 2014b) et ensuite ajusté avec un jeu de données *réel*. Il utilise un algorithme SGD standard, un momentum de

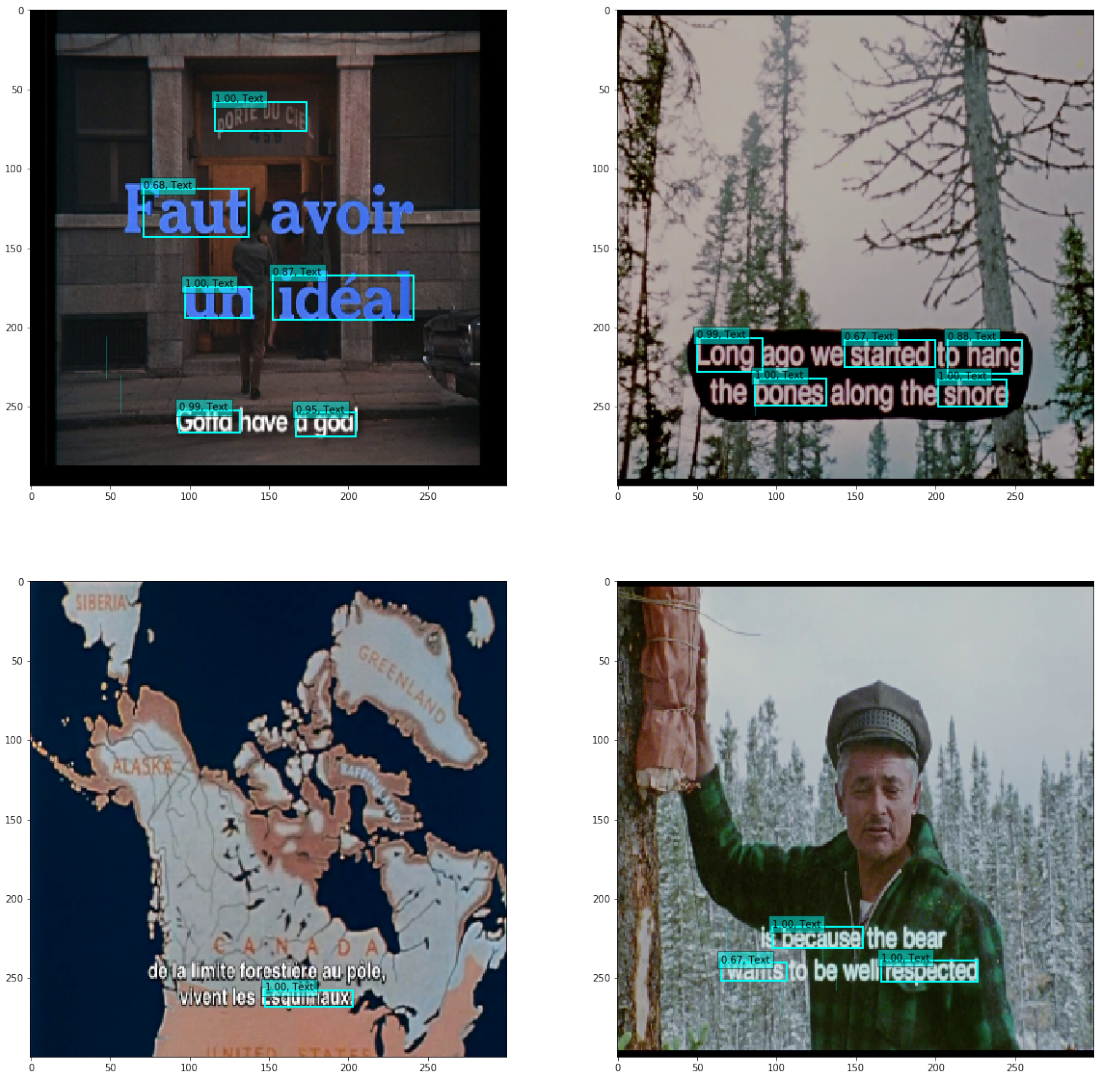

Figure 4.1: Exemples de résultats avec l'approche basée sur TextBoxes

0.9 et les images sont augmentées lors de l'entraînement. Le taux d'apprentissage est à 10^{-3} pour les 60k premières itérations et ensuite à 10^{-4} pour les 30k suivants. Finalement, Seglink utilise deux variables α pour le seuil des segments et β pour le seuil des liens. Il faut donc trouver la configuration optimale grâce à un algorithme de *gridsearch* (LaValle *et al.*, 2004) qui cherche les meilleurs paramètres possible.

De plus, les auteurs mentionnent que leur protocole d'évaluation change selon leurs jeux de données : ils changent α et β pour chaque jeu de données pour qu'ils s'adaptent mieux à celui-ci. De façon analogue à TextBoxes, nous tentons de réimplémenter SegLink avec Keras et Tensorflow.

Après avoir entraîné le modèle avec les mêmes jeux de données de l'article, nous obtenons des résultats similaires aux auteurs sur ICDAR 2013 avec une F-mesure de 85,3% et 75% sur ICDAR 2015. Toutefois, quand nous les comparons à notre jeu de données ONF-Text, nous obtenons, avec le *gridsearch*, un seuil α de 0,4, β de 0,1 et une F-Mesure de 36,59%. Nous sommes loin d'obtenir des résultats similaires aux expériences citées dans l'article. Cela est probablement dû au fait que le modèle n'arrive pas à généraliser sur des images plus difficiles ou différentes que celles utilisées durant leur entraînement.

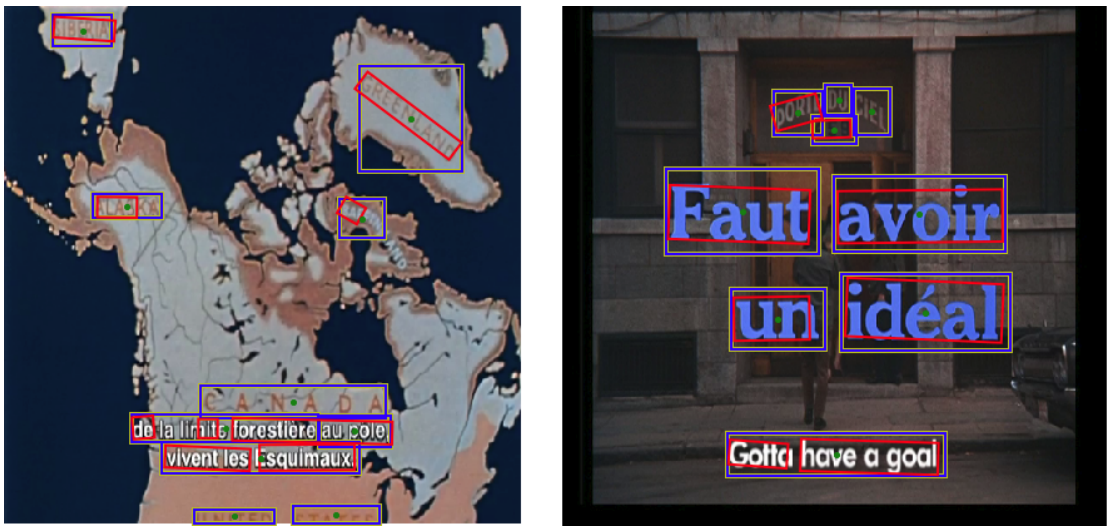

Figure 4.2: Exemples de résultats avec l'approche basée sur Seglink

Par contre, sur le corpus ONF-Text, lorsqu'on regarde les résultats de manière qualitative sur les images (voir la figure 4.2), nous notons une amélioration, là où le texte est incurvé ou assez grand, par rapport à TextBoxes.

Des mots incurvés ou obstrués sont maintenant détectés, mais nous notons aussi une augmentation des faux positifs (167) et de faux négatifs (86), 46% de précision et 30% de rappel. Voilà pourquoi la F-Mesure est si mauvaise sur nos données.

4.3.3 Approche basée sur Textboxes++ (Liao *et al.*, 2018)

TextBoxes++, comme nous l'avons mentionné dans l'état de l'art, est une nouvelle itération qui supporte le texte orienté et qui est entraînée avec une nouvelle fonction de coût qui tient compte de la reconnaissance de caractères.

Nous tentons encore une fois de l'implémenter avec Keras et Tensorflow selon les spécifications de l'article, mais nous abandonnons toutefois l'idée de refaire la fonction de coût.

Notre solution est entraînée sur 5 jeux de données, soit ICDAR 2015, COCO-Text, SVT et ICDAR 2013 en 3 étapes. Premièrement, le modèle est pré entraîné 60k fois sur SynthText pour tous les jeux de test. Ensuite, à la deuxième étape, il est entraîné sur les images d'entraînement de chaque jeu de données. Finalement, la dernière étape entraîne des images plus grandes et avec un taux d'apprentissage plus grand. Notre expérimentation à été réalisée avec un serveur Z820 ayant une carte GeForce GTX 1080 Ti.

Les résultats de notre implémentation sont similaires à l'article, nous obtenons une F-Mesure de 92,4 % sur SynthText. Toutefois, les résultats sont décevants avec les données de l'ONF. Nous n'obtenons qu'une F-Mesure de 24,8 %. Et lorsque nous analysons qualitativement les résultats (voir les exemples à la figure 4.3), nous remarquons que même si le modèle est plus performant sur les textes orientés ou incurvés, il semble avoir de la difficulté sur les textes ayant un mauvais contraste ou même, étonnamment avec le texte en noir et blanc.

Figure 4.3: Exemples de résultats sur Textboxes++

4.3.4 Approche basée sur Real-time Scene Text Detection with Differentiable Binarization (DB) (Liao *et al.*, 2019)

Pour cette expérimentation, nous utilisons la librairie, libre de droit, *PaddleOCR* (Du *et al.*, 2020) qui se veut une ressource pour créer des outils OCR pratiques et multilingues et qui favorise les meilleures pratiques dans le domaine de la détection et de la reconnaissance de texte. Au moment d'écrire ces lignes, *PaddleOCR*

implémente trois modèles de détection de texte, dont DB (Liao *et al.*, 2019) et SAST (Wang *et al.*, 2019).

Comme nous l'avons mentionné dans l'état de l'art, DB utilise des méthodes de segmentation et une fonction de binarisation dérivable.

Pour réaliser notre expérimentation, nous avons donc dû adapter le jeu de données ONF-Text à *PaddleOCR* avec l'outil d'annotation PPOCRLabel. Il s'agit d'un outil très pratique puisqu'il fait une première phase de détection et de reconnaissance. Ensuite, l'utilisateur n'a qu'à corriger et ajouter les informations manquantes.

Étant donné que le modèle pré entraîné est disponible en téléchargement, nous n'avons eu qu'à l'évaluer avec les différents modèles fournis avec l'outil. Rappelons que ces modèles figurent, au moment d'écrire ces lignes, au sommet dans les classements sur le jeu de données ICDAR 2015 et Total-Text.

Les résultats sont étonnants : nous obtenons une F-Mesure nettement moins bonne qu'avec TextBoxes++ et Seglink. DB obtient 16,1% de F-mesure, 28,1% de rappel et 11,3% de précision sur ONF-Text. Toutefois, quand nous regardons les résultats de façon qualitative sur les exemples de résultats (4.4), les résultats semblent meilleurs qu'avec les autres solutions : il réussit à détecter du texte tronqué, incurvé, obstrué et ayant un mauvais contraste.

On remarque que le modèle détecte mal le texte quand il est grand et qu'il y a beaucoup d'espace entre les caractères. Par exemple, pour le texte « pour vivre sa vie » ou pour « CANADA » (4.4). De plus, il ne semble pas détecter complètement le texte à la verticale, comme dans l'affiche de Coca-Cola.

Bien que ce modèle soit un des plus performants à ce jour, pourquoi obtient-on une mauvaise F-Mesure sur les données de ONF-Text ? Quelles sont les différences

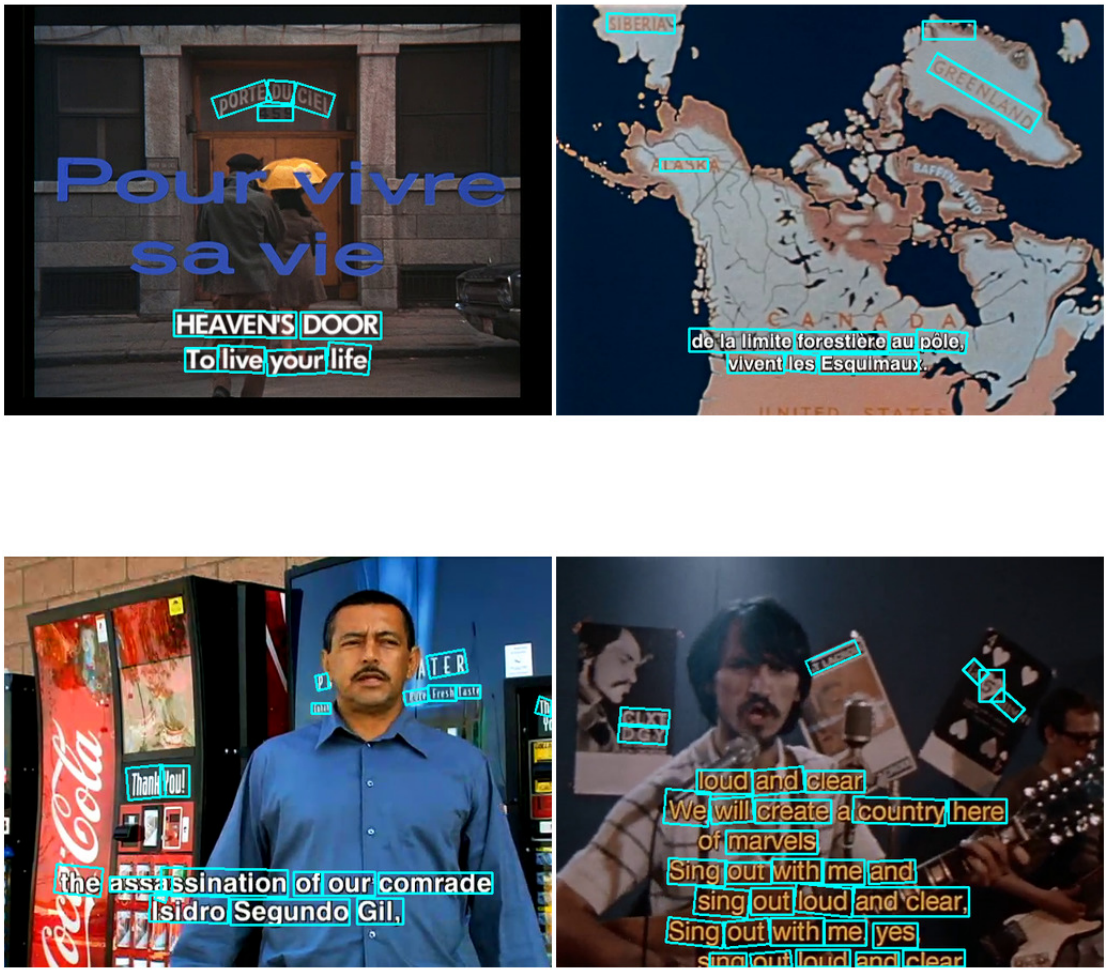

Figure 4.4: Exemple de résultats sur ONF-Text avec DB

avec les modèles précédents ?

Pour mieux comprendre, nous avons ré annoté notre jeu de donnée (ONF-Text) avec des polygones à la place de boîtes. DB supporte le texte avec plusieurs orientations et les polygones permettent d'être plus précis dans l'encadrement des zones de texte. Néanmoins, l'analyse qualitative a démontré que DB conservait certaines lacunes.

Tableau 4.1: Résultat des tests réalisés avec les différents modèles de détection de texte que nous avons implémenté. Dans le tableau, R est le rappel, P la précision et F la F-mesure. À la fin, on compare la rapidité d'exécution sur les images de ONF-Text.

Modèle (Détection)	**ICDAR 2013**			**ICDAR 2015**			**Total-Text**			**ONF-Text**			**Temps/s**
	R	**P**	**F**	**R**	**P**	**F**	**R**	**P**	**F**	**R**	**P**	**F**	
TextBoxes	88.0	82.0	85.0	-	-	-	-	-	-	61	40.4	50.7	0.73
SegLink	83.0	87.7	85.3	76.8	73.1	75.0	-	-	-	30	46	36.6	20.5
TextBoxes++ MultiScale	86	92	89	78.5	87.8	82.9				34.6	19.3	24.8	9.6
DB (ResNet-50)	-	-	-	83.2	91.8	87.3	82.5	87.1	84.7	28.1	11.3	16.1	62
SAST (ResNet-50)	-	-	-	83.7	91.4	87.4	78.4	89.6	83.6	30.7	11	16.24	23.63
TextFuseNet (ResNet-101)	92.3	96.5	94.3	89.7	94.7	92.1	85.3	89.0	87.1	-	-	-	3.7

Pour valider que le modèle fonctionnait bien, nous l'avons toutefois évalué avec plusieurs autres jeux de données (ICD, 2015; El Abed *et al.*, 2011; Ch'ng *et al.*, 2020) et nous avons obtenu des résultats similaires, même meilleurs que ceux de l'article Liao *et al.* (2019). Nous pouvons ainsi en déduire que ce sont nos données qui présentent des difficultés pour le modèle.

4.3.5 Résultats des expérimentations de la détection de texte

Nous présentons les résultats que nous avons obtenus dans les différentes expérimentations sur la détection de texte dans le tableau 4.1. Dans le tableau, **F** représente la F-Mesure, **R** le rappel et **P** la précision. De plus, tous les résultats sont des pourcentages.

En conclusion, en analysant les résultats, toutes les solutions modernes offrent des résultats similaires. Notre choix doit donc se baser sur la façon dont le modèle réagit dans des cas difficiles, sa rapidité, la qualité de la solution et sa facilité d'intégration.

Nous avons fait plusieurs itérations, mais ce n'est que récemment que l'équipe de Baidu *PaddleOCR* (Du *et al.*, 2020) propose une solution libre de droits et suivants les meilleurs standards dans le domaine. Elle est aussi réimplémentée en

C++, ce qui la rend très rapide.

Lors de notre expérimentation, nous avons ré-implémenté Textboxes, Seglink, TextBoxes++, mais nous avons aussi essayé la solution originale des auteurs, ou d'autres réimplémentations. Nous avons rapidement observé des problèmes de performance lors du traitement des images. Qui plus est, est-ce optimal de gérer les modèles, qui sont destinés à évoluer très rapidement, en les ré implémentant à chaque fois ?

Dans cette optique, les modèles fournis par *PaddleOCR* sont toujours les plus récents et la solution en C++ obtient une vitesse largement supérieure aux solutions en Python. En particulier pour l'algorithme de *Non Maximum Suppression* qui est très vorace en complexité mémoire et temporelle. Il s'agit d'une merveilleuse librairie pour la recherche car elle force l'adoption d'un cadre de travail favorisant ainsi les bonnes pratiques.

Nous avons donc choisi Real-time Scene Text Detection with Differentiable Binarization de Liao *et al.* (2019). Malgré les résultats assez médiocres sur nos données, DB offre généralement de meilleurs résultats sur l'ensemble des autres jeux de données et est significativement plus rapide que les autres solutions avec 62 fps. Nous rappellerons au lecteur que nos données sont particulièrement difficiles à traiter lorsque l'on rencontre des cas difficiles à l'ONF et qu'il est normal d'avoir de mauvais résultats.

4.4 Reconnaissance de texte

Après la détection vient la reconnaissance. Ici la tâche est moins ardue, car, comme nous l'avons expliqué dans l'état de l'art, les modèles modernes sont très avancés et c'est pour cela que nous avons concentré nos efforts dans notre enquête sur la détection de texte. Dans l'état de l'art, nous mentionnons que CRNN est largement

suffisant pour faire l'OCR. Malgré tout, nous avons pris soin d'expérimenter les solutions pour valider les scores.

Pour ce faire, nous avons annoté manuellement le jeu de données ONF-Text pour l'adapter au format des modèles OCR grace à LabelImg (Tzutalin, 2015) comme nous l'avons décrit dans le chapitre sur les données 3.4.

	jeux de données	
Modèles	ICDAR15	ONF-Text
CRNN Bi-LSTM ResNet-34	82,7 %	**32,4 %**
SRN Resnet-50	**88,5 %**	30,7 %
Star-Net Resnet34	84,4 %	10,5 %

Tableau 4.2: Résultats (F-Mesure) des tests de reconnaissance de texte

Ainsi, nous confirmons notre hypothèse que le modèle CRNN de Shi *et al.* (2015) est robuste et obtient une meilleure F-Mesure pour notre jeu de données ONF-Text (Tableau 4.2) et que les nouvelles solutions n'apportent finalement rien de mieux compte tenu de leur complexité (Yu *et al.*, 2020) et du manque d'amélioration sur nos données (voir chapitre 2).

4.5 Post-correction de la reconnaissance

Nous avons bien montré que la détection et la reconnaissance de texte dans les images sont un problème non résolu. Il est commun qu'on détecte du texte partiellement ou que la reconnaissance fait erreur sur quelques lettres (par exemple, *patt* à la place de *path*). Ces erreurs se répètent d'un modèle à l'autre et d'une reconnaissance à l'autre et la post-correction tente de remédier à cela.

Un des avantages de la reconnaissance dans les vidéos est que le texte se répète souvent sur plusieurs plans dans le temps. Nous pouvons ainsi regrouper ces textes et en extraire les résultats ayant le plus haut taux de confiance.

Nous avons donc développé **postocr**, une solution de post-correction de reconnaissance de texte.

Elle prend en entrée le fichier de sortie de la détection et reconnaissance de texte et corrige selon certaines heuristiques pour obtenir la phrase ayant le plus haut taux de confiance possible. Par exemple, l'OCR confond souvent les « n » avec des « h » ou inversement. **postocr** nettoie les faux positifs, par exemple des mots de moins de 3 lettres complètement isolés du reste du texte.

Une autre tâche de **postocr** est de rassembler temporellement le texte : pour un certain laps de temps, le même texte apparaît à l'image. Par exemple, un sous-titre doit rester au moins quelques secondes à l'image, pendant ce temps l'arrière-plan peut changer, améliorant ainsi le contraste. Ce rassemblement permet donc la création de fichiers de sous-titre, l'indexation selon le temps d'apparition et l'optimisation du taux de succès de la détection, car selon les plans, les taux de succès changent.

Pour regrouper le texte, **postocr** utilise un calcul de ratio alliant la distance temporelle entre les textes reconnus et la similarité des textes. La similarité est calculée grâce à l'algorithme de similarité cosinus (Singhal, 2001). Une fois la similarité obtenue, nous multiplions celle-ci selon la distance temporelle : plus elle est courte, plus la similarité est grande et plus la distance est grande, plus la similarité est petite.

Une fois regroupés, nous pouvons maintenant traiter le texte pour maximiser le taux de succès dans chaque petit groupe de texte. Si un mot est mieux détecté et reconnu dans un plan plutôt que dans l'autre, nous utiliserons celui-ci. Pour s'assurer qu'il s'agit du même mot, nous comparons les coordonnées des mots dans l'image et s'il s'agit à peu près des mêmes coordonnées, ce sont les mêmes mots.

4.6 Analyse et indexation

La dernière étape est l'analyse et l'indexation, on utilise le moteur de recherche et d'analyse **Elasticsearch** (Gormley et Tong, 2015) basé sur Lucene (`http://lucene.apache.org`). Lucene est une puissante plateforme qui permet l'indexation, la correction de texte, l'analyse, et autre opération, de texte. Cette étape est cruciale, car elle permet de pallier les erreurs que le post-traitement OCR n'a pas pu corriger.

Ainsi, avant d'indexer il est primordial de faire l'analyse du texte. Cela consiste à décomposer le texte en unités qu'on nomme termes. Pour créer ces termes, il faut extraire les mots, enlever les mots inutiles, nettoyer la ponctuation, filtrer les caractères spéciaux, en les remplaçant par leurs équivalents, passer aux minuscules. Ensuite, nous utilisons un stemmeur qui réduit les mots à leur racine.

Lowercase : Transforme le texte en minuscule.

Stop : Enlève les mots qui comme « le, la, il, etc. », car ils sont considérés comme du bruit.

asciifolding : remplace tous les caractères spéciaux par de l'ASCII.

Elision : Enlève les articles des mots tels que les « l', t' ».

Snowball : Stemmeur qui extrait la racine des mots pour ne conserver que l'essentiel.

Il ne reste plus qu'à spécifier un schéma Elasticsearch où l'on peut lier cette analyse à notre structure de données, soit essentiellement, du texte et le temps où il s'affiche dans le film. Elasticsearch s'occupe ensuite d'appeler Lucene pour créer les index.

4.6.1 Recherche d'information

La dernière partie de notre solution se trouve dans la recherche d'information et la présentation des résultats. Elasticsearch utilise l'algorithme de similarité BM25 Robertson et Zaragoza (2009) et supporte les agrégations, un mécanisme qui permet d'effectuer des statistiques sur les données pour par exemple faire un top 10 des mots utilisés dans un film avec le total correspondant, de faire une répartition des mots les plus utilisés selon les années de production, etc. Ces informations sont particulièrement utiles pour afficher des filtres de recherche à l'utilisateur : film en langue "française" de type "fiction" entre les années "1930 et 1935" produit par l'ONF.

Lorsqu'on fait des requêtes dans un moteur de recherche, il faut faire un compromis entre le rappel et la précision. Si nous retournons tous les documents, nous aurons un rappel de 100%, mais ce n'est pas utile. Si nous sommes trop stricts, nous aurons une précision de 100%, mais trop peu de documents seront retournés. Parfois, il vaut mieux obtenir plusieurs résultats et les élaguer par des filtres. Dans d'autres cas, il vaut mieux n'obtenir que très peu de résultats, mais qu'ils soient corrects.

Listing 1: Analyseur pour la recherche

Dans cette requête (voir code dans la tableau Listing 1), de manière analogue à l'analyse des données, nous nettoyons le texte et nous l'analysons avec un stemmeur. Il ne reste alors que l'essentiel du texte recherché, ce qui rend la requête plus générale.

Nous favorisons donc l'obtention du plus grand nombre de résultats possibles, ce qui permet de supporter les coquilles et les erreurs de reconnaissance. Bref, l'indexation permet à la recherche une plus grande robustesse. Toutefois, grâce aux agrégations d'Elasticsearch, il est facile d'élaguer les résultats.

Il est dès lors possible de faire, par exemple, une recherche sur le terme « Orignal », et ES nous retournera toutes les occurrences de ce mot dans les séquences de films que nous avons indexées.

4.7 Solution choisie

Toutes ces recherches nous amènent enfin vers notre véritable contribution : permettre aux usagers de rechercher du texte dans les films de la collection de l'ONF. Dans cette partie, nous allons présenter l'architecture de haut niveau de cette solution et faire un retour sur l'utilisation de celle-ci.

Architecture de haut niveau

À l'ONF, nous utilisons déjà une architecture de traitement des médias. Ainsi, la reconnaissance était déjà vouée à y être intégrée. Toutefois, elle représente quelques nouveaux défis, par exemple, comment archiver ces données de reconnaissance ? Devrait-on les régénérer ? Lorsque de nouveaux modèles émergent, devrait-on rouler à nouveau la reconnaissance sur tous les films ou seulement les nouveaux ?

Dans l'architecture (figure 4.5), il était important de laisser ces portes entre-ouvertes, car nous ne pouvions pas prendre de décision claire dans l'immédiat.

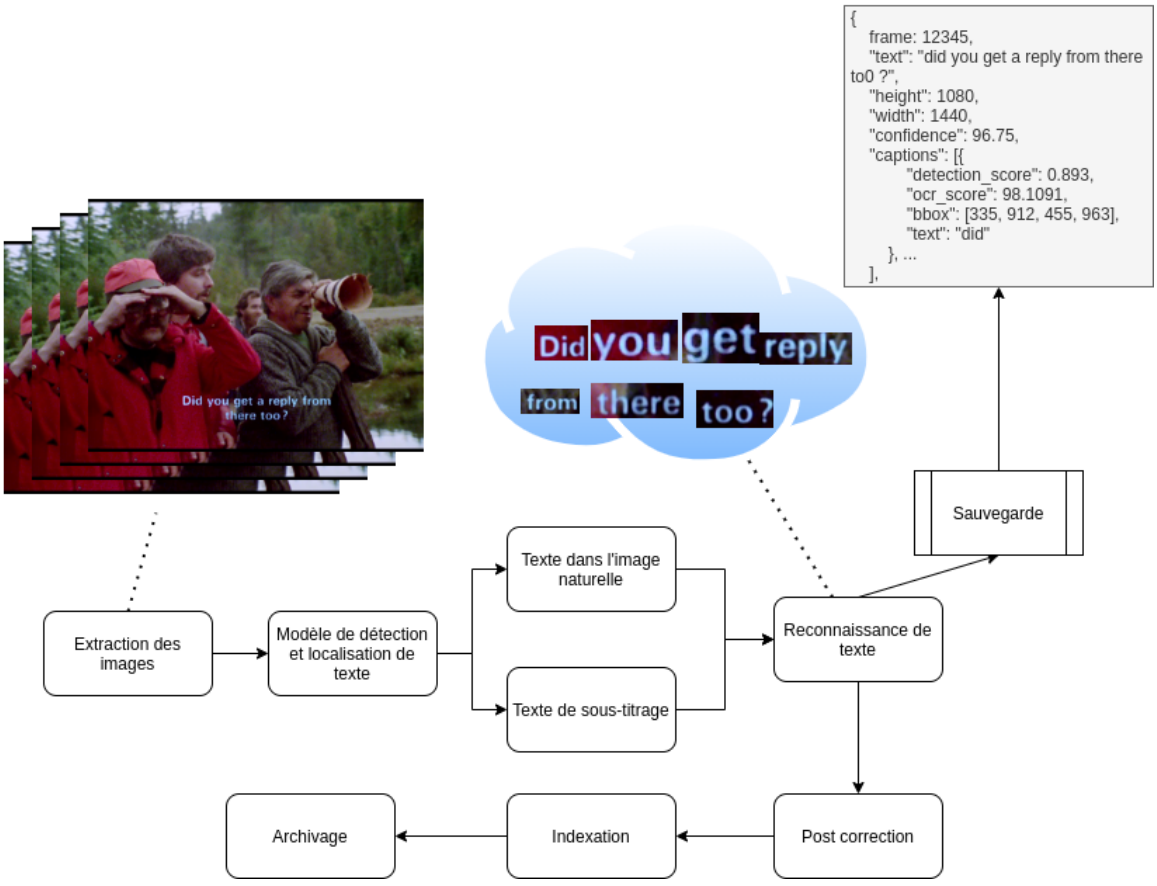

Figure 4.5: Architecture de haut niveau de la reconnaissance de texte dans les films.

De plus, advenant une urgence ou une perte des données, nous pourrions re générer les index, car nous sauvegardons les données brutes de la reconnaissance.

Statistiques

Aujourd'hui, grâce à notre service, nous pouvons traiter un film en quasi temps réel, mais en ne traitant que 1/10 des images extraites. À ce jour, nous avons traité environ 1200 films que nous avons identifiés comme intéressants préalablement. Maintenant que nous avons éprouvé la solution, nous planifions de traiter l'ensemble des 14 000 films de la collection.

Media asset management : MAM

À l'ONF, depuis la mise en place du plan de numérisation en 2008, nous avons créé un gestionnaire d'oeuvres multimédia qui est la vitrine de notre collection. Avec cet outil, les usagers peuvent faire des recherches sur la collection, visionner les films ou consulter des informations techniques sur ceux-ci.

Cet outil s'interface déjà à plusieurs index Elasticsearch, c'était donc l'endroit parfait pour accueillir et rendre disponible ces nouvelles données comme nous pouvons le remarquer dans l'exemple de la figure 4.6.

Figure 4.6: Capture d'écran de la recherche textuelle dans le MAM

CONCLUSION

Dans ce mémoire, pour atteindre notre objectif de pouvoir faire des recherches sur le contenu textuel dans les films de l'ONF, nous avons commencé par explorer les techniques de vision par ordinateur et nous avons ensuite exploré les algorithmes d'apprentissage machine pour détecter le texte. Après avoir rencontré une impasse, nous avons cheminé vers l'apprentissage profond. Pour développer nos expérimentations et mesurer notre progression, nous avons créé et annoté notre propre jeu de données : ONF-Text.

Nous avons présenté les différentes architectures de détection d'objets, nous avons découvert que ces architectures étaient similaires à celles de la détection de texte dans les images. Nous avons exploré les architectures de reconnaissance de texte. Nous avons expérimenté plusieurs modèles et obtenu des résultats semblables à l'état de l'art (Liao *et al.*, 2019, 2018; Shi *et al.*, 2017; Ye *et al.*, 2020) dans la détection de texte. Nous avons ensuite établi que les solutions de reconnaissance de texte CRNN Shi *et al.* (2015) obtenaient toujours les meilleurs résultats pour nos données. Finalement, nous avons créé une librairie qui traite les résultats de détection et reconnaissance et en extrait l'essentiel corrigé selon des heuristiques et algorithmes de traitement automatique du langage naturel.

Après avoir exploré et testé les meilleures solutions pour traiter la collection de 14 000 films de l'Office national du film du Canada, ce mémoire a permis d'innover en créant une plateforme qui détecte, reconnaît, collige, corrige, indexe et rend disponible l'information textuelle des films de l'ONF via notre intranet (MAM).

Aujourd'hui, en quelques secondes nous pouvons chercher et trouver l'ensemble des scènes où le mot « ours » ou « Trudeau » est écrit, ceci pouvait prendre des mois auparavant. Nous pouvons donc dire que nous avons atteint nos objectifs, et même plus, car nous avons aussi amélioré l'accessibilité et la découvrabilité des oeuvres de notre patrimoine cinématographique.

Toutefois, nous avons aussi vu les limitations de ces modèles. Certains ne détectent pas le texte vertical (Liao *et al.*, 2019, 2018), d'autres pas le texte dans le texte (Ye *et al.*, 2020). Nous avons remarqué que la reconnaissance faisait souvent les mêmes erreurs et nous avons tenté de les corriger(Hämäläinen et Hengchen, 2019). Nous avons découvert que corriger ces erreurs provoquait parfois d'autres erreurs. Aujourd'hui, en écrivant ces lignes, il est clair que certains problèmes abordés dans ce mémoire restent irrésolus et nécessitent encore beaucoup de travail.

Une piste d'amélioration serait d'expérimenter avec des réseaux comme *Temporally Identity-Aware SSD with Attentional LSTM* (Chen *et al.*, 2018) qui joignent SSD et RNN. Nous pourrions adapter son architecture de façon analogue à Textboxes++ pour traiter le texte dans les films et le RNN permettrait de traiter les données temporelles et ainsi d'obtenir potentiellement de meilleurs résultats.

De plus, dans l'état de l'art, nous avons effleuré la post-correction de la reconnaissance de texte, mais derrière ce petit sous-chapitre se cache un vaste domaine de recherche. En 2019, l'ICDAR a créé pour la deuxième fois une compétition sur le sujet et plusieurs compétiteurs ont créé des solutions innovantes se basant par exemple sur BERT (Devlin *et al.*, 2018). Ces solutions de modélisation du langage sont très complexes et dépassent largement le domaine de cette recherche.

Nous avons aussi abordé les solutions de détection d'objets, mais les solutions de détection de texte tendent aujourd'hui à s'inspirer des modèles de segmentation d'image comme DB (Liao *et al.*, 2019) et SAST (Wang *et al.*, 2019).

Finalement, ce mémoire montre combien il est important de continuer la recherche en intelligence artificielle dans des organismes publics tels que l'ONF, car ultimement, elle permet aux Canadien.ne.s de découvrir leur culture.

RÉFÉRENCES

(2015). *ICDAR '15 : Proceedings of the 2015 13th International Conference on Document Analysis and Recognition (ICDAR)*, USA. IEEE Computer Society.

Aaron Courville, Y. B. e. a. (2019). Reproducibility in machine learning. Récupéré de https://iclr.cc/Conferences/2019/Schedule?showEvent=635

Abadi, M., Agarwal, A. et et al., P. B. (2015). *TensorFlow : Large-Scale Machine Learning on Heterogeneous Systems*. [Logiciel. En ligne]. Récupéré de https://www.tensorflow.org/

Amidi, A. et Amidi, S. (2021a). *CS 230 - Deep Learning*. Récupéré de https://stanford.edu/~shervine/teaching/cs-230/cheatsheet-convolutional-neural-networks

Amidi, A. et Amidi, S. (2021b). *CS 230 - Deep Learning*. Récupéré de https://stanford.edu/~shervine/teaching/cs-230/cheatsheet-convolutional-neural-networks

Awasthi, A., Sarawagi, S., Goyal, R., Ghosh, S. et Piratla, V. (2019). Parallel iterative edit models for local sequence transduction. *CoRR*, *abs/1910.02893*. Récupéré de http://arxiv.org/abs/1910.02893

Babenko, B., Belongie, S. et Wang, K. (2011). End-to-end scene text recognition. Dans *2011 IEEE International Conference on Computer Vision (ICCV 2011)*,

1457–1464., Los Alamitos, CA, USA. IEEE Computer Society. http://dx.doi.org/10.1109/ICCV.2011.6126402

Bengio, Y., Ducharme, R., Vincent, P. et Janvin, C. (2003). A neural probabilistic language model. *J. Mach. Learn. Res.*, *3*, 1137–1155. Récupéré de http://dl.acm.org/citation.cfm?id=944919.944966

Bochkovskiy, A., Wang, C. et Liao, H. M. (2020). Yolov4 : Optimal speed and accuracy of object detection. *CoRR*, *abs/2004.10934*. Récupéré de https://arxiv.org/abs/2004.10934

Bojanowski, P., Grave, E., Joulin, A. et Mikolov, T. (2017). Enriching word vectors with subword information. *Transactions of the Association for Computational Linguistics*, *5*, 135–146.

Borisyuk, F., Gordo, A. et Sivakumar, V. (2018). Rosetta : Large scale system for text detection and recognition in images. 71–79. http://dx.doi.org/10.1145/3219819.3219861

Chen, X., Wu, Z. et Yu, J. (2018). TSSD : temporal single-shot object detection based on attention-aware LSTM. *CoRR*, *abs/1803.00197*. Récupéré de http://arxiv.org/abs/1803.00197

Ch'ng, C., Chan, C. S. et Liu, C. (2020). Total-text : toward orientation robustness in scene text detection. *Int. J. Document Anal. Recognit.*, *23*(1), 31–52.

Chollet, F. *et al.* (2015). *Keras*. [Logiciel. En ligne]. Récupéré de https://github.com/fchollet/keras

Coates, A., Carpenter, B., Case, C., Satheesh, S., Suresh, B., Wang, T., Wu, D. J. et Ng, A. Y. (2011). Text detection and character recognition in scene images with unsupervised feature learning. Dans *Proceedings of the 2011 International*

Conference on Document Analysis and Recognition, 440–445. IEEE Computer Society.

Cortes, C. et Vapnik, V. (1995). Support-vector networks. *Machine learning*, *20*(3), 273–297.

Devlin, J., Chang, M., Lee, K. et Toutanova, K. (2018). BERT : pre-training of deep bidirectional transformers for language understanding. *CoRR*, *abs/1810.04805*. Récupéré de http://arxiv.org/abs/1810.04805

Dollar, P., Appel, R., Belongie, S. et Perona, P. (2014). Fast feature pyramids for object detection. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, *36*, 1532–1545. http://dx.doi.org/10.1109/TPAMI.2014.2300479

Du, Y., Li, C., Guo, R., Yin, X., Liu, W., Zhou, J., Bai, Y., Yu, Z., Yang, Y., Dang, Q. et Wang, H. (2020). PP-OCR : A practical ultra lightweight OCR system. *CoRR*, *abs/2009.09941*. Récupéré de https://arxiv.org/abs/2009.09941

El Abed, H., Liu, W. et Märgner, V. (2011). International conference on document analysis and recognition (icdar 2011) - competitions overview. Dans *2013 12th International Conference on Document Analysis and Recognition*, 1437–1443. IEEE Computer Society. http://dx.doi.org/10.1109/ICDAR.2011.286

Everingham, M., Gool, L. V., Williams, C. K. I., Winn, J. M. et Zisserman, A. (2009). The pascal visual object classes (voc) challenge. *International Journal of Computer Vision*, *88*, 303–338.

Fournier, J. (2019). *La numérisation et la préservation dans le monde numérique des diverses collections audiovisuelles à l'ONF*. Récupéré de https://www.crkn-rcdr.ca/sites/crkn/files/2019-11/Sharing-Digital-Heritage_8c_Fournier_FR.pdf

Fukushima, K. (1980). Neocognitron : A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological Cybernetics*, *36*, 193–202.

Girshick, R. B. (2015). Fast R-CNN. *CoRR*, *abs/1504.08083*. Récupéré de http://arxiv.org/abs/1504.08083

Girshick, R. B., Donahue, J., Darrell, T. et Malik, J. (2013). Rich feature hierarchies for accurate object detection and semantic segmentation. *CoRR*, *abs/1311.2524*. Récupéré de http://arxiv.org/abs/1311.2524

Glorot, X. et Bengio, Y. (2010). Understanding the difficulty of training deep feedforward neural networks. Dans Y. W. Teh et M. Titterington (dir.). *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, volume 9 de *Proceedings of Machine Learning Research*, 249–256., Chia Laguna Resort, Sardinia, Italy. PMLR. Récupéré de https://proceedings.mlr.press/v9/glorot10a.html

Gómez, L. et Karatzas, D. (2015). *Scene Text Recognition : No Country for Old Men ?*, Dans *Computer Vision - ACCV 2014 Workshops : Singapore, Singapore, November 1-2, 2014, Revised Selected Papers, Part II*, (p. 157–168). Springer International Publishing.

Goodfellow, I., Bengio, Y. et Courville, A. (2016). *Deep Learning*. MIT Press. http://www.deeplearningbook.org.

Google. (2021). *Machine Learning Crash Course*. Récupéré de https://developers.google.com/machine-learning/crash-course

Gormley, C. et Tong, Z. (2015). *Elasticsearch : The Definitive Guide*. O'Reilly Media, Inc.

Graves, A., Fernández, S., Liwicki, M., Bunke, H. et Schmidhuber, J. (2007). Unconstrained online handwriting recognition with recurrent neural networks. Dans *Proceedings of the 20th International Conference on Neural Information Processing Systems*, NIPS'07, 577–584.

Gupta, A., Vedaldi, A. et Zisserman, A. (2016). Synthetic data for text localisation in natural images. *CoRR*, *abs/1604.06646*, 2315–2324.

Hämäläinen, M. et Hengchen, S. (2019). From the paft to the fiiture : a fully automatic NMT and word embeddings method for OCR post-correction. Dans *Proceedings of the International Conference on Recent Advances in Natural Language Processing (RANLP 2019)*, 431–436., Varna, Bulgaria. INCOMA Ltd. http://dx.doi.org/10.26615/978-954-452-056-4_051

Hinton, G. (2015). *Ask me anything Geoffrey Hinton*. Récupéré de https://www.reddit.com/r/MachineLearning/comments/2lmo0l/ama_geoffrey_hinton/

Hochreiter, S. et Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, *9*(8), 1735–1780.

Hornik, K. (1991). Approximation capabilities of multilayer feedforward networks. *Neural Networks*, *4*(2), 251–257.

Horowitz, S. L. et Pavlidis, T. (1976). Picture segmentation by a tree traversal algorithm. *J. ACM*, *23*(2), 368–388.

Huang, W., Qiao, Y. et Tang, X. (2014). *Robust Scene Text Detection with Convolution Neural Network Induced MSER Trees*, Dans *Computer Vision – ECCV 2014 : 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part IV*, (p. 497–511). Springer International Publishing.

Hubel, D. H. et Wiesel, T. N. (1962). Receptive fields, binocular interaction and functional architecture in the cat's visual cortex. *The Journal of Physiology*, *160*.

Ian Hogarth, N. (2020). *State of AI Report 2020*. Récupéré de https://www.stateof.ai/

Iwamura, M., Morimoto, N., Tainaka, K., Bazazian, D., Gomez, L. et Karatzas, D. (2017). ICDAR2017 robust reading challenge on omnidirectional video. Dans *Proc. IAPR Int. Conf. on Doc. Anal. and Recognit. (*ICDAR*)*, volume 1, 1448–1453.

Jaderberg, M., Simonyan, K., Vedaldi, A. et Zisserman, A. (2014a). Reading text in the wild with convolutional neural networks. *CoRR*, *abs/1412.1842*.

Jaderberg, M., Simonyan, K., Vedaldi, A. et Zisserman, A. (2014b). Synthetic data and artificial neural networks for natural scene text recognition. *CoRR*, *abs/1406.2227*. Récupéré de http://arxiv.org/abs/1406.2227

Jain, L. C. et Medsker, L. R. (1999). *Recurrent Neural Networks : Design and Applications*. USA : CRC Press, Inc.

Jawahar, C. V., Alahari, K. et Mishra, A. (2012). Top-down and bottom-up cues for scene text recognition. Dans *2012 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2687–2694. IEEE Computer Society. http://dx.doi.org/10.1109/CVPR.2012.6247990

Jia, Y. (2013). *Caffe : An Open Source Convolutional Architecture for Fast Feature Embedding*. [Logiciel. En ligne]. Récupéré de http://caffe.berkeleyvision.org/

Jocher, G. (2020). *ultralytics/yolov5 : v3.1 - Bug Fixes and Performance Improvements*. Zenodo. http://dx.doi.org/10.5281/zenodo.4154370

Karatzas, D., Shafait, F., Uchida, S., Iwamura, M., Bigorda, L. G. i., Mestre, S. R., Mas, J., Mota, D. F., Almazàn, J. A. et de las Heras, L. P. (2013a). Icdar 2013 robust reading competition. p. 1484–1493. IEEE Computer Society. http://dx.doi.org/10.1109/ICDAR.2013.221

Karatzas, D., Shafait, F., Uchida, S., Iwamura, M., i Bigorda, L. G., Mestre, S. R., Mas, J., Mota, D. F., Almazan, J. A. et de las Heras, L. P. (2013b). Icdar 2013 robust reading competition. Dans *2013 12th International Conference on Document Analysis and Recognition (ICDAR)*, 1484–1493. IEEE, IEEE Computer Society.

Kay, A. (2007). Tesseract : An open-source optical character recognition engine. *Linux J.*, *2007*(159), 2.

Khan, T., Sarkar, R. et Mollah, A. (2021). Deep learning approaches to scene text detection : a comprehensive review. *Artificial Intelligence Review*, *54*, 1–60. http://dx.doi.org/10.1007/s10462-020-09930-6

Kozen, D. C. (1992). *Depth-First and Breadth-First Search*, Dans *The Design and Analysis of Algorithms*, (p. 19–24). Springer New York

Larochelle, H. (2009). *Études de techniques d'appentissage non-supervisé pour l'amélioration de l'entraînement supervisé de modèles connexionnistes*. (PhD dissertation). Université de Montréal.

LaValle, S. M., Branicky, M. S. et Lindemann, S. R. (2004). On the relationship between classical grid search and probabilistic roadmaps. *The International Journal of Robotics Research*, *23*(7-8), 673–692.

LeCun, Y. A., Bottou, L., Orr, G. B. et Müller, K.-R. (2012). *Efficient BackProp*, (p. 9–48). Springer Berlin Heidelberg : Berlin, Heidelberg

Liao, M., Shi, B. et Bai, X. (2018). TextBoxes++ : A single-shot oriented scene text detector. *IEEE Transactions on Image Processing*, *27*(8), 3676–3690. http://dx.doi.org/10.1109/TIP.2018.2825107

Liao, M., Shi, B., Bai, X., Wang, X. et Liu, W. (2017). Textboxes : A fast text detector with a single deep neural network. Dans *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence, February 4-9, 2017, San Francisco, California, USA.*, 4161–4167.

Liao, M., Wan, Z., Yao, C., Chen, K. et Bai, X. (2019). Real-time scene text detection with differentiable binarization. *CoRR*, *abs/1911.08947*. Récupéré de http://arxiv.org/abs/1911.08947

Lienhart, R. (1997). Automatic text recognition for video indexing. Dans *Proceedings of the fourth ACM international conference on Multimedia*, 11–20. ACM, University of Mannheim.

Lin, T., Goyal, P., Girshick, R. B., He, K. et Dollár, P. (2017). Focal loss for dense object detection. *CoRR*, *abs/1708.02002*. Récupéré de http://arxiv.org/abs/1708.02002

Lin, T., Maire, M., Belongie, S. J., Bourdev, L. D., Girshick, R. B., Hays, J., Perona, P., Ramanan, D., Dollár, P. et Zitnick, C. L. (2014). Microsoft COCO : common objects in context. *CoRR*, *abs/1405.0312*. Récupéré de http://arxiv.org/abs/1405.0312

Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S. E., Fu, C. et Berg, A. C. (2015). SSD : single shot multibox detector. *CoRR*, *abs/1512.02325*.

Lloyd, S. (1982). Least squares quantization in pcm. *IEEE Transactions on Information Theory*, *28*(2), 129–137.

Lucas, S. M., Panaretos, A., Sosa, L., Tang, A., Wong, S. et Young, R. (2003). ICDAR 2003 Robust Reading Competitions. Dans *Proceedings of the Seventh International Conference on Document Analysis and Recognition*, 682–687. IEEE Computer Society. http://dx.doi.org/10.1109/ICDAR.2003.1227749

Manning, C. D. et Schütze, H. (1999). *Foundations of Statistical Natural Language Processing*. Cambridge, Massachusetts : The MIT Press. Récupéré de http://nlp.stanford.edu/fsnlp/

Massa, F. et Girshick, R. (2018). *maskrcnn-benchmark : Fast, modular reference implementation of Instance Segmentation and Object Detection algorithms in PyTorch*. Récupéré de https://github.com/facebookresearch/maskrcnn-benchmark

Matas, J., Chum, O., Urban, M. et Pajdla, T. (2004). Robust wide baseline stereo from maximally stable extremal regions. *Image and Vision Computing*, *22*, 761–767.

Merkel, D. (2014). Docker : lightweight linux containers for consistent development and deployment. *Linux journal*, *2014*(239), 2.

Mikolov, T., Sutskever, I., Chen, K., Corrado, G. et Dean, J. (2013). Distributed representations of words and phrases and their compositionality. *CoRR*, *abs/1310.4546*. Récupéré de http://arxiv.org/abs/1310.4546

Miller, G., Beckwith, R., Fellbaum, C., Gross, D. et Miller, K. (1990). Introduction to wordnet : An on-line lexical database. *International Journal of Lexicography*, *3*, 235–244.

Neumann, L. (2017). *Scene Text Localization and Recognition in Images and Videos*. (Thèse de doctorat). Faculty of the Electrical Engineering of the

Czech Technical University in Prague. Récupéré de http://cmp.felk.cvut.cz/~neumalu1/LukasNeumann_Disertation.pdf

Neumann, L. et Matas, J. (2012). Real-time scene text localization and recognition. Dans *2012 IEEE Conference on Computer Vision and Pattern Recognition*, 3538–3545. IEEE, IEEE Computer Society.

OpenCV (2015). Open source computer vision library. [Logiciel. En ligne]. Récupéré de https://github.com/opencv/opencv

Padilla, R., Netto, S. L. et da Silva, E. A. B. (2020). A survey on performance metrics for object-detection algorithms. Dans *2020 International Conference on Systems, Signals and Image Processing (IWSSIP)*, 237–242. http://dx.doi.org/10.1109/IWSSIP48289.2020.9145130

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V. *et al.* (2011). Scikit-learn : Machine learning in python. *Journal of machine learning research*, *12*(Oct), 2825–2830.

Pennington, J., Socher, R. et Manning, C. (2014). Glove : Global vectors for word representation. Dans *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 1532–1543., Doha, Qatar. Association for Computational Linguistics. http://dx.doi.org/10.3115/v1/D14-1162

Peters, M. E., Neumann, M., Iyyer, M., Gardner, M., Clark, C., Lee, K. et Zettlemoyer, L. (2018). Deep contextualized word representations. *CoRR*, *abs/1802.05365*. Récupéré de http://arxiv.org/abs/1802.05365

Phan, T. Q., Shivakumara, P., Tian, S. et Tan, C. L. (2013). Recognizing text with perspective distortion in natural scenes. Dans *IEEE International Conference*

on Computer Vision, ICCV 2013, Sydney, Australia, December 1-8, 2013, 569–576. IEEE Computer Society. http://dx.doi.org/10.1109/ICCV.2013.76

Porter, M. (1980). An algorithm for suffix stripping. *Program*, *40*, 211–218.

Ramachandran, P., Zoph, B. et Le, Q. V. (2017). Searching for activation functions. *CoRR*, *abs/1710.05941*. Récupéré de http://arxiv.org/abs/1710.05941

Redmon, J., Divvala, S. K., Girshick, R. B. et Farhadi, A. (2015). You only look once : Unified, real-time object detection. *CoRR*, *abs/1506.02640*.

Redmon, J. et Farhadi, A. (2016). YOLO9000 : better, faster, stronger. *CoRR*, *abs/1612.08242*. Récupéré de http://arxiv.org/abs/1612.08242

Redmon, J. et Farhadi, A. (2018). Yolov3 : An incremental improvement. *CoRR*, *abs/1804.02767*. Récupéré de http://arxiv.org/abs/1804.02767

Ren, S., He, K., Girshick, R. B. et Sun, J. (2015). Faster R-CNN : towards real-time object detection with region proposal networks. *CoRR*, *abs/1506.01497*. Récupéré de http://arxiv.org/abs/1506.01497

Robertson, S. et Zaragoza, H. (2009). The probabilistic relevance framework : Bm25 and beyond. *Foundations and Trends® in Information Retrieval*, *3*(4), 333–389. http://dx.doi.org/10.1561/1500000019

Rosenblatt, F. (1958). The perceptron : a probabilistic model for information storage and organization in the brain. *Psychological review*, *65 6*, 386–408.

Rosenfeld, A. et Thurston, M. (1971). Edge and curve detection for visual scene analysis. *IEEE Transactions on Computers*, *C-20*(5), 562–569. http://dx.doi.org/10.1109/T-C.1971.223290

Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M. S., Berg, A. C. et Li, F. (2014). Imagenet large scale visual recognition challenge. *CoRR*, *abs/1409.0575*. Récupéré de http://arxiv.org/abs/1409.0575

Schapire, R. (2013). Explaining adaboost. In *Empirical inference* 37–52. Springer.

Shephard, D. (2021). *A Detailed Study and Recent Research on OCR*. Récupéré de https://history-computer.com/inventions/gismo-of-david-shepard/

Shi, B., Bai, X. et Belongie, S. J. (2017). Detecting oriented text in natural images by linking segments. *CoRR*, *abs/1703.06520*.

Shi, B., Bai, X. et Yao, C. (2015). An end-to-end trainable neural network for image-based sequence recognition and its application to scene text recognition. *CoRR*, *abs/1507.05717*.

Shruthi, P. et Verma, D. (2021). *A Detailed Study and Recent Research on OCR*. Zenodo. http://dx.doi.org/10.1109/ARTCom.2009.45

Singhal, A. (2001). Modern information retrieval : A brief overview. *IEEE Data Eng. Bull.*, *24*, 35–43.

Swartz, C. S. (2005). *Understanding Digital Cinema : A Professional Handbook*. Focal Press.

Tomar, S. (2006). Converting video formats with ffmpeg. *Linux Journal*, *2006*(146), 10.

Tzutalin. (2015). *LabelImg*. [Logiciel. En ligne]. Git code (2015). Récupéré de https://github.com/tzutalin/labelImg

Uijlings, J., Sande, K., Gevers, T. et Smeulders, A. (2013). Selective search for object recognition. *International Journal of Computer Vision*, *104*, 154–171. http://dx.doi.org/10.1007/s11263-013-0620-5

Veit, A., Matera, T., Neumann, L., Matas, J. et Belongie, S. J. (2016). Coco-text : Dataset and benchmark for text detection and recognition in natural images. *CoRR*, *abs/1601.07140*. Récupéré de http://arxiv.org/abs/1601.07140

Wallach, H., Larochelle, H., Beygelzimer, A., F., Alché-Buc, Fox, E. et Garnett, R. (dir.) (2019). *PyTorch : An Imperative Style, High-Performance Deep Learning Library*. Curran Associates, Inc. Récupéré de http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf

Wang, P., Zhang, C., Qi, F., Huang, Z., En, M., Han, J., Liu, J., Ding, E. et Shi, G. (2019). A single-shot arbitrarily-shaped text detector based on context attended multi-task learning. *CoRR*, *abs/1908.05498*. Récupéré de http://arxiv.org/abs/1908.05498

Wang, Y. et Lian, Z. (2020). Exploring font-independent features for scene text recognition. Dans C. W. Chen, R. Cucchiara, X. Hua, G. Qi, E. Ricci, Z. Zhang, et R. Zimmermann (dir.). *MM 20 : The 28th ACM International Conference on Multimedia, Virtual Event / Seattle, WA, USA, October 12-16, 2020*, 1900–1920. ACM.

Wikimedia Foundation. (2013). *Exemple d'un perceptron — Wikipedia, The Free Encyclopedia*. Récupéré de https://commons.wikimedia.org/wiki/File:Perceptron_moj.png

Wikimedia Foundation. (2021). *Sous-titrage — Wikipedia, The Free Encyclopedia*. [Online ; accessed 29-Jan-2022]. Récupéré de https://fr.wikipedia.org/wiki/Sous-titrage

Williams, T. L. et Li, R. (2016). Advanced image classification using wavelets and convolutional neural networks. Dans *15th IEEE International Conference on Machine Learning and Applications, ICMLA 2016, Anaheim, CA, USA, December 18-20, 2016*, 233–239. IEEE Computer Society.

Wolf, C. et Jolion, J.-M. (2006). Object count/area graphs for the evaluation of object detection and segmentation algorithms. *Int. J. Doc. Anal. Recognit.*, *8*(4), 280–296.

Wu, Y., Kirillov, A., Massa, F., Lo, W.-Y. et Girshick, R. (2019). *Detectron2*. Récupéré de https://github.com/facebookresearch/detectron2

Yang, W., Zhang, H. et Lin, J. (2019). Simple applications of BERT for ad hoc document retrieval. *CoRR*, *abs/1903.10972*. Récupéré de http://arxiv.org/abs/1903.10972

Ye, J., Chen, Z., Liu, J. et Du, B. (2020). Textfusenet : Scene text detection with richer fused features. Dans C. Bessiere (dir.). *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI 2020*, 516–522. ijcai.org.

Yin, X.-C., Zuo, Z.-Y., Tian, S. et Liu, C.-L. (2016). Text detection, tracking and recognition in video : a comprehensive survey. *IEEE Transactions on Image Processing*, *25*(6), 2752–2773.

Yu, D., Li, X., Zhang, C., Han, J., Liu, J. et Ding, E. (2020). Towards accurate scene text recognition with semantic reasoning networks. *arXiv preprint arXiv :2003.12294*.

Yu, D., Wang, H., Chen, P. et Wei, Z. (2014). Mixed pooling for convolutional neural networks. Dans D. Miao, W. Pedrycz, D. Slezak, G. Peters, Q. Hu, et R. Wang (dir.). *Rough Sets and Knowledge Technology - 9th International*

Conference, RSKT 2014, Shanghai, China, October 24-26, 2014, Proceedings, volume 8818 de *Lecture Notes in Computer Science*, 364–375. Springer.

Zaidi, S. S. A., Ansari, M. S., Aslam, A., Kanwal, N., Asghar, M. N. et Lee, B. (2021). A survey of modern deep learning based object detection models. *CoRR*, *abs/2104.11892*. Récupéré de https://arxiv.org/abs/2104.11892

Zeiler, M. D. et Fergus, R. (2013). Stochastic pooling for regularization of deep convolutional neural networks. *arXiv preprint arXiv :1301.3557*.

Zitnick, C. L. et Dollár, P. (2014). Edge boxes : Locating object proposals from edges. Dans D. Fleet, T. Pajdla, B. Schiele, et T. Tuytelaars (dir.). *Computer Vision – ECCV 2014*, 391–405. Springer International Publishing.