Architectures de Transformer légères pour la reconnaissance de textes manuscrits anciens

Killian Barrere

COLLEGE MATHS, TELECOMS
DOCTORAL INFORMATIQUE, SIGNAL
BRETAGNE SYSTEMES, ELECTRONIQUE

INSA INSTITUT NATIONAL DES SCIENCES APPLIQUÉES RENNES

THÈSE DE DOCTORAT DE

L'INSTITUT NATIONAL DES SCIENCES APPLIQUÉES DE RENNES

ÉCOLE DOCTORALE N° 601
Mathématiques, Télécommunications, Informatique, Signal, Systèmes, Électronique
Spécialité : *Informatique*

Par
Killian Barrere

Architectures de Transformer légères pour la reconnaissance de textes manuscrits anciens

Thèse présentée et soutenue à Rennes, le 20 décembre 2023
Unité de recherche : IRISA
Thèse N° : 23ISAR 45 / D23 - 45

Rapporteurs avant soutenance :

Christophe GARCIA — Professeur des universités, INSA Lyon
Thierry PAQUET — Professeur des universités, Université de Rouen Normandie

Composition du Jury :

Président :	Harold MOUCHÈRE	Professeur des universités, Université de Nantes
Examinateurs :	Christophe GARCIA	Professeur des universités, INSA Lyon
	Thierry PAQUET	Professeur des universités, Université de Rouen Normandie
	Josep LLADÓS	Professeur associé, Universitat Autónoma de Barcelona, Espagne
	Harold MOUCHÈRE	Professeur des universités, Université de Nantes
Dir. de thèse :	Bertrand COÜASNON	Maître de conférences HDR, INSA Rennes
Co-dir. de thèse :	Aurélie LEMAITRE	Maître de conférences HDR, Université Rennes 2

Invité :

Co-encadr. de thèse : Yann SOULLARD — Maître de conférences, Université Rennes 2

Remerciements

Je tiens d'abord à adresser des remerciements tout particulièrement à mes encadrants de thèse, sans qui je ne serais probablement pas arrivé jusque-là. Merci à Yann pour ses nombreux conseils prodigués, ses nombreuses relectures détaillées, ainsi que tout le temps qu'il a pu m'accorder. Merci aussi à Aurélie qui a su m'aider à m'orienter durant ce périple qu'est la thèse, m'aider avec de nombreuses relectures, ou encore pour quelques pauses autour de jeux de société. Enfin, merci à Bertrand pour le temps qu'il m'a accordé, et pour ses nombreux conseils avisés qui m'ont permis d'améliorer la qualité du manuscrit.

Ma reconnaissance va aussi en direction des rapporteurs et des membres du jury de thèse : Christophe, Thierry, Josep ainsi qu'Harold. Je les remercie pour avoir accepté de relire ce manuscrit, d'assister à la soutenance, mais aussi pour les échanges très intéressants qui ont eu lieu.

Un grand merci à tous mes collègues de l'équipe SHADoc avec qui j'ai grandement apprécié travailler pour de nombreuses raisons. Cela inclut les différents moments de travail, de pause, de détente après le travail, ou encore des moments musicaux. Merci encore pour la bonne ambiance et le soutien qu'ils m'ont apporté, sans tout cela, je ne serai probablement pas arrivé jusqu'à cette étape aussi sereinement. Je tiens aussi à remercier plus particulièrement Florent I., Bruno ainsi que Frédéric pour avoir assuré du côté de la régie en parallèle de ma soutenance.

Merci aussi à Solène Tarride pour le travail conjoint réalisé sur la partie génération de données synthétiques, et en particulier ses apports sur la partie dégradation pour des documents anciens. Je la remercie également pour tous les aspects évoqués au paragraphe précédent.

Je tiens à remercier les différentes personnes qui m'ont aidé dans la dernière ligne droite à relire et à corriger les différents chapitres de ce manuscrit. Merci à ma mère, Jadden, Michaël, Raphaël, Florent M., Martin V., Orens, mon père, Élise, Jonathan, et à nouveau, merci à mes encadrants de thèse.

Je remercie également mes amis et ma famille qui m'ont soutenu tout le long de la thèse, que ce soit de manière directe ou indirecte. Je tiens en particulier à remercier ma mère qui m'a grandement aidé dans la dernière ligne droite.

J'adresse aussi mes remerciements aux différents élèves de l'INSA Rennes ou de l'ENS Rennes à qui j'ai grandement apprécié donner des cours. Ils se sont toujours montrés très sympathiques et joyeux. C'est aussi grâce à eux que j'apprécie autant donné des cours, et que ces cours sont une véritable bouffée d'air frais qui m'a permis d'avancer dans ma recherche dans la bonne humeur.

Ces travaux ont bénéficié d'un accès aux moyens de calcul de l'IDRIS au travers des allocations de ressources 2020-AD011011591 et 2021-AD011012550 attribuées par GENCI, que je tiens également à remercier pour la possibilité d'utiliser ces moyens de calcul, mais aussi pour la grande qualité de service que j'ai pu constater tout au long de ma thèse.

Sommaire

Liste des acronymes

— **BLSTM** : Couche de mémoire court et long terme bidirectionnelle (*Bidirectional Long Short-Term Memory*)

— **CER** : Taux d'erreur caractère (*Character Error Rate*)

— **CFTNN** : Notre architecture convolutive à Transformer rapide (*Convolutional Fast Transformer Neural Network*)

— **CRNN** : Réseau de neurones convolutif et récurrent (*Convolutional Recurrent Neural Network*) (ou encore CNN-LSTM)

— **CTC** : Approche de classification temporelle connexioniste (*Connectionist Temporal Classification*)

— **FCN** : Réseau totalement convolutif (*Fully Convolutional Network*)

— **GAN** : Réseau de neurones génératif (*Generative Adversarial Network*)

— **LM** : Modèle de langue (*Language Model*)

— **LSTM** : Couche de mémoire court et long terme (*Long Short-Term Memory*)

— **LT** : Notre Architecture de Transformer légère (*Lightweight Transformer*)

— **MLT** : Notre architecture de Transformer autorégressive légère et modulable (*Modular Lightweight Transformer*), modulable en VLT et LT

— **MT** : Notre architecture de Transformer de taille modérée (*Medium Transformer*), c'est une version non légère de MLT

— **ReLU** : *Rectified Linear Unit*

— **seq2seq** : Architecture séquence à séquence

— **VLT** : Notre architecture de Transformer très légère (*Very Lightweight Transformer*)

— **WER** : Taux d'erreur mot (*Word Error Rate*)

Introduction

Contexte général de la thèse

Les systèmes de reconnaissance automatique d'écriture visent à transcrire des écritures du monde réel dans un format numérique. Il est attendu de ces systèmes d'être à la fois plus rapide et moins coûteux que des annotateurs humains. La transcription d'écriture a de nombreux intérêts, comme par exemple la reconnaissance et la traduction en temps réel d'écriture ou la requête d'informations. Les systèmes de reconnaissance automatique d'écriture sont particulièrement utiles pour des archives ou bibliothèques. En effet, de nombreux documents sont déjà numérisés ou prêts à être numérisés dans un souci de conservation. Obtenir les transcriptions de leur contenu est ainsi particulièrement utile pour pouvoir classifier et requêter rapidement le contenu. Ces documents peuvent par la même occasion être mis à la disposition du public, par exemple via une interface web.

La reconnaissance de texte manuscrit est un problème difficile. La forme des caractères manuscrits est fortement variable, que ce soit dû à des variations inter-scripteurs ou intra-scripteur. D'autres variations s'ajoutent aussi selon l'époque des documents considérés, comme l'utilisation de ligatures. Les écritures modernes sont souvent homogènes, et il est en général simple de disposer de données annotées (c'est-à-dire des images et leur transcription) pour entraîner les systèmes de reconnaissance. Les documents anciens (du XV^{e} siècle au XIX^{e} siècle) sont quant à eux hétérogènes et sujets à de nombreuses dégradations causées par le temps (détérioration du papier, effacement de l'encre, *etc.*). De plus, l'évolution de la langue au cours du temps, peut venir ajouter une difficulté supplémentaire. L'écriture est alors particulièrement complexe à reconnaître, même pour un expert humain. Le nombre de documents annotés est donc en général faible. La figure 1 montre par exemple des images de lignes de texte provenant de différents documents.

Deux types d'approches se distinguent pour reconnaître l'écriture. Les approches dites "en ligne" utilisent la dynamique du geste du tracé, qui peut être obtenue à partir d'une tablette ou d'un stylet numérique. Cependant, ces informations ne sont pas souvent disponibles pour de l'écriture récente, et inexistantes pour de l'écriture ancienne. Il est donc nécessaire d'utiliser des approches "hors-ligne", qui se focalisent sur l'analyse des pixels

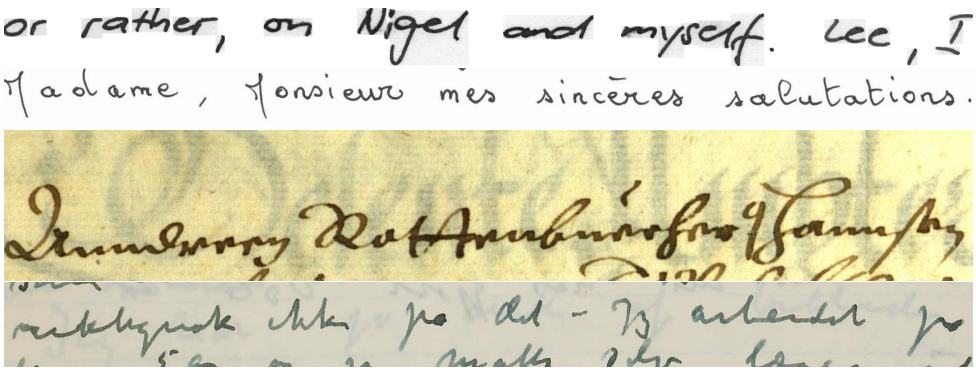


FIGURE 1 – Exemples de lignes de texte aux styles moderne et ancien. Les images proviennent dans l'ordre des jeux de données IAM [MB02], RIMES [Aug+06], READ de l'ICFHR 2016 [San+16] et READ de l'ICFHR 2018 [Str+18].

dans l'image pour reconnaître l'écriture. Les systèmes classiques de reconnaissance hors-ligne sont en général basés sur trois étapes (illustrées dans la figure 2) :

1. une analyse structurelle du document, notamment afin d'extraire des lignes de texte isolées ;
2. une reconnaissance de l'écriture à partir des images de lignes de texte ;
3. ainsi qu'une étape de corrections linguistiques en utilisant des modèles de langue.

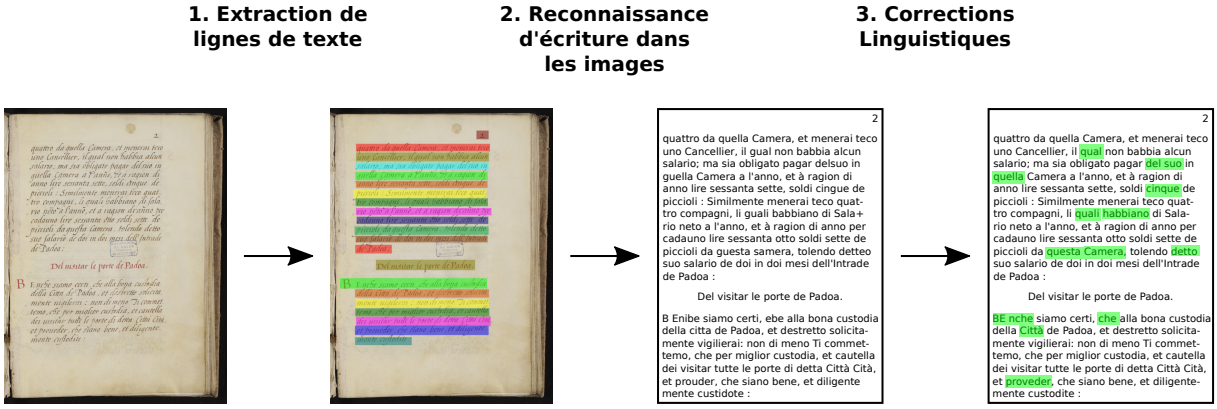


FIGURE 2 – Les différentes étapes d'un système de reconnaissance d'écriture hors-ligne complet.

Dans ce manuscrit, nous traitons la problématique de reconnaissance d'écriture manuscrite avec des approches hors-ligne. Plus précisément, nous nous intéressons à la reconnaissance de lignes de textes isolées. Nous étudions à la fois des documents modernes, mais aussi des documents anciens bien plus complexes en comparaison. Nous n'abordons en revanche pas l'étape d'analyse de la structure des documents.

Difficultés existantes dans les systèmes de reconnaissance d'écriture manuscrite

Les méthodes basées sur des réseaux de neurones sont aujourd'hui utilisées pour résoudre de nombreuses tâches dans différents domaines. Grâce à des poids entraînables, elles sont capables d'apprendre de manière automatique à résoudre des tâches complexes. Cependant, des volumes importants de données annotées sont requis pour obtenir des résultats satisfaisants. Dans le domaine de la reconnaissance de texte manuscrit, les réseaux de neurones ont montré de faibles taux d'erreur ces dernières années et représentent la quasi-totalité des méthodes utilisées.

Ces dernières années, les réseaux de neurones convolutifs et récurrents (CRNN) ont été largement utilisés [DZN16 ; SBY16 ; SDN16 ; BM17 ; Pui17 ; WYL17 ; Dut+18 ; Sou+19]. Plus récemment, des architectures basées uniquement sur des couches de convolution (FCN) ont été proposées [Coq+19 ; Ing+19 ; Pen+19 ; Ptu+19 ; YHM20 ; YB20 ; CCP21 ; CCP22]. Ces architectures présentent diverses limites. D'une part, les architectures CRNN ont des temps d'apprentissage longs en raison de l'absence de parallélisation, tandis que la mémorisation du contexte peut parfois être insuffisante. D'autre part les architectures FCN sont limitées dans la modélisation d'informations contextuelles larges.

Les architectures type Transformer [Vas+17] ont récemment été créées comme une alternative efficace aux couches de récurrence pour résoudre des tâches de traitement automatique de la langue [Dev+18]. Elles se basent sur l'utilisation d'un mécanisme d'attention pour porter de l'attention sur des caractéristiques jugées pertinentes. Elles sont capables d'intégrer un contexte bien plus large que celui offert par les réseaux récurrents, et peuvent corriger des erreurs grâce à des capacités de modélisation de la langue. Enfin, elles tirent parti du parallélisme offert par les cartes graphiques modernes, et permettent un entraînement bien plus efficace.

Très récemment, elles ont été appliquées dans le domaine de la reconnaissance d'écriture manuscrite, et sont, à ce jour, très fortement utilisées [SK21 ; Kan+22 ; WZG22 ; WZG21 ; dAr+22 ; Ria+22 ; CCP23 ; Li+23]. Elles tirent notamment profit d'architectures de plus en plus larges pour obtenir de bons résultats.

Grâce à leurs propriétés, les architectures Transformer semblent intéressantes pour reconnaître des écritures complexes dans des documents anciens. Leur aptitude à modéliser la langue bénéficierait à la reconnaissance de langues complexes, anciennes et variées au

cours du temps. De plus, leur capacité à apprendre des contextes larges peut aider à la prise de décisions lors de la reconnaissance des caractères.

En revanche, des architectures de Transformer larges nécessitent de plus en plus de données annotées pour être entraînées efficacement. Or, en reconnaissance d'écriture, peu de données annotées sont généralement disponibles. C'est d'autant plus le cas pour les documents anciens qui sont difficiles et très coûteux à annoter. Certaines approches comblent le manque de données annotées en utilisant des jeux de données supplémentaires, publics ou privés. Mais ces données ne sont pas toujours disponibles, en lien avec le document considéré, ou faciles à obtenir. La génération de données synthétiques pour entraîner les modèles est alors une solution. Cependant, la génération de données annotées adaptées aux documents anciens n'est pas une tâche aisée compte tenu de la diversité et de la difficulté de ces documents.

De fait, et à notre connaissance, les approches type Transformer n'ont pas été appliquées pour la tâche de reconnaissance d'écriture sur des documents anciens.

Contributions de la thèse

Nous cherchons à améliorer la reconnaissance d'écriture dans des documents anciens en utilisant des architectures de type Transformer. En effet, les propriétés de ces architectures, dont nous discuterons ultérieurement, nous semblent particulièrement adaptées aux documents anciens.

Les modèles type Transformer de plus en plus large ne sont, selon nous, pas adaptés avec les faibles quantités de données annotées souvent inhérentes à l'écriture manuscrite, et plus particulièrement pour de l'écriture ancienne. Il nous semble important de trouver un compromis sur la taille pour maximiser les performances d'un modèle, étant donnée une quantité de données annotées.

Dans le cadre des documents anciens, nous choisissons de proposer des architectures de Transformer légères en nombre de paramètres. Une architecture légère nécessite moins de données annotées et moins de temps d'entraînement, en comparaison à une architecture large. Une architecture légère est alors moins coûteuse, ce qui permet de réaliser des économies en termes de temps, d'argent ainsi que d'énergie. De plus, nous verrons qu'un modèle léger peut permettre des prédictions bien plus rapides, ce qui peut avoir son importance dans certains scénarios. Néanmoins, une architecture Transformer, bien que légère, n'en reste pas moins complexe à entraîner sur de telles données. Nous propo-

sons ainsi plusieurs stratégies additionnelles pour entraîner efficacement nos architectures, comme par exemple l'utilisation de données synthétiques.

Les contributions de cette thèse peuvent être résumées de la manière suivante :

— **des architectures de Transformer légères** adaptées à la reconnaissance d'écriture avec peu de données annotées ;

— **des algorithmes de génération de données synthétiques paramétrables**, afin de générer des données synthétiques adaptées aux documents modernes ou anciens ;

— **une stratégie d'entraînement et de spécialisation** sur des documents spécifiques avec peu de données annotées ;

— **une stratégie de prédiction basée sur de l'augmentation en test et un algorithme de vote** afin de réduire les erreurs commises par les modèles.

De plus, dans le but de permettre la reproduction de nos résultats, ainsi que la production de futurs résultats, nous mettons à disposition l'entièreté du code utilisé, des paramètres choisis, ainsi que les poids des modèles entraînés.

Bien que nous cherchons à reconnaître des documents anciens, nos architectures et approches ne sont pas limitées aux documents anciens. Nous verrons qu'elles sont aussi adaptées à la reconnaissance d'écriture moderne. Les algorithmes de générations de données synthétiques proposés étant paramétrables, ils peuvent être utilisés pour générer des données synthétiques au style visuel demandé. Enfin, les stratégies d'entraînement et de prédiction proposées ne se limitent pas à nos approches et peuvent bénéficier à différentes architectures existantes.

Dans ce manuscrit, nous montrerons que nos architectures de Transformer légères sont capables d'être entraînées avec peu de données annotées. Sur des documents modernes et sans données additionnelles, elles obtiennent des résultats au niveau de l'état de l'art. Nous verrons que la génération de données synthétiques ainsi que les diverses stratégies proposées se montrent grandement utiles, et en particulier dans le cadre de documents anciens. Elles permettent en particulier à nos approches d'obtenir des résultats compétitifs, voire meilleurs que certaines architectures plus larges. Sur le jeu de données READ 2018 [Str+18], nous obtenons en particulier les meilleurs résultats en n'utilisant uniquement 16 pages annotées (soit environ 500 lignes) pour spécialiser un modèle, ainsi que sur certains documents complexes, et ceci malgré l'utilisation d'une architecture Transformer complexe. En particulier, nos modèles légers peuvent être utilisés pour prédire jusqu'à l'équivalent de 60 pages de lignes de texte par seconde.

Contenu du manuscrit

Le manuscrit de thèse est organisé de la manière suivante :

Chapitre 1 - Formalisation du problème de reconnaissance d'écriture manuscrite Nous commençons par proposer une formalisation du problème de reconnaissance d'écriture, et nous introduisons les différentes notations que nous utilisons dans le reste du manuscrit.

Partie I - État de l'art Nous étudions en détail les différentes méthodes existantes. Dans le chapitre 2 nous présentons les différentes architectures de réseaux de neurones utilisées ces dernières années. Le chapitre 3 présente les architectures de Transformer dans le domaine du traitement automatique de la langue, tandis que le chapitre 4 présente leur utilisation dans le domaine de la reconnaissance de texte manuscrit. Le chapitre 5 aborde les problématiques liées aux données dans le domaine, ainsi que les différentes approches qui traitent ce problème.

Partie II - Contributions Nous détaillons les différentes contributions de la thèse. Le chapitre 6 est consacré à une première architecture de Transformer légère, basée sur l'utilisation d'un encodeur Transformer, et qui a l'avantage d'être à la fois légère et rapide. Dans le chapitre 7, nous présentons une architecture légère de Transformer utilisant un décodeur Transformer pour modéliser la langue. De plus, nous proposons deux variantes pour s'adapter aux quantités de données annotées. Le chapitre 8 présente ensuite nos algorithmes de générations de données synthétiques. Nous abordons enfin la stratégie d'entraînement et de spécialisation avec peu de données annotées ainsi que la stratégie de prédiction dans le chapitre 9.

Partie III - Résultats expérimentaux Dans le chapitre 10, nous validons nos approches et les différentes contributions de la thèse sur des documents modernes. Dans le chapitre 11, nous évaluons ensuite nos approches sur des données anciennes et complexes, dans des scénarios réalistes avec différentes quantités de données annotées disponibles. Nos résultats sont comparés avec les approches à l'état de l'art, et nous montrons que nos architectures sont capables d'obtenir de meilleurs résultats, tout en étant capable de s'entraîner avec peu de données annotées.

Conclusion générale Enfin, nous concluons le manuscrit en résumant les différentes contributions et résultats obtenus, tout en proposant diverses perspectives.

Chapitre 1

Formalisation du problème de reconnaissance d'écriture manuscrite

Le problème de reconnaissance d'écriture que nous traitons vise à fournir une transcription associée à l'image d'une ligne de texte d'écriture. L'écriture peut être vue comme une séquence de par son ordre de lecture, et les systèmes de reconnaissance traitent généralement l'écriture, ou l'image de l'écriture, en utilisant cette séquentialité.

Dans ce chapitre, nous explicitons les différents formalismes utilisés dans ce manuscrit.

1.1 Notations générales

Une séquence de vecteurs $\boldsymbol{S} \in \mathbb{R}^{N \times D}$, est une séquence de longueur N contenant des vecteurs dans $\mathbb{R}^D$. De manière analogue, $\boldsymbol{S}$ peut être vue comme une matrice dans $\mathcal{M}_{N,D}(\mathbb{R})$.

On notera $\boldsymbol{S}_i \in \mathbb{R}^D$ pour désigner le i-ème vecteur de la séquence de vecteurs $\boldsymbol{S}$, avec $i \in [\![1; N]\!]$. En particulier $\boldsymbol{S}_i$ peut être vu comme la ligne i de la matrice associée à la séquence de vecteurs $\boldsymbol{S}$.

Enfin, $\boldsymbol{S}_{i,j} \in \mathbb{R}$ désignera le j-ème coefficient du vecteur $\boldsymbol{S}_i$, de manière équivalente à la notation matricielle classique.

1.2 Formalisation de l'apprentissage

Étant donné un ensemble de données annotées $\mathcal{D}$, on note $\boldsymbol{X}$ une image de lignes de texte et $\boldsymbol{y}$ sa transcription associée (ou vérité terrain). Plus précisément, $\boldsymbol{X}$ contient l'ensemble de l'information de l'image, à savoir les valeurs des pixels de l'image. La transcription $\boldsymbol{y} = (\boldsymbol{y_1}, ..., \boldsymbol{y_{|y|}})$ est une séquence de caractères, dans un alphabet Σ, telle que

$\boldsymbol{y} \in \Sigma^*$. La tâche de reconnaissance d'écriture vise à apprendre une fonction de *mapping* F associant à une image de ligne de texte $\boldsymbol{X}$ une prédiction $\hat{\boldsymbol{y}}$ telle que :

$$\hat{\boldsymbol{y}} = F(\boldsymbol{X}) \tag{1.1}$$

Il est à noter que $\hat{\boldsymbol{y}}$ et $\boldsymbol{y}$ peuvent être de natures différentes. La plupart des approches considèrent à la place une représentation de $\hat{\boldsymbol{y}}$ sous la forme d'une séquence de probabilité de caractères. C'est en particulier le cas des approches que nous proposons dans ce manuscrit. Pour comparer ces objets, une opération dans les fonctions de coût est en général utilisée. Par la suite, nous supposerons que $\hat{\boldsymbol{y}}$ et $\boldsymbol{y}$ sont comparables.

On cherchera à minimiser la différence entre la prédiction $\hat{\boldsymbol{y}}$ et la vérité terrain $\boldsymbol{y}$. Pour évaluer et quantifier cette différence, une fonction de coût $\mathcal{L}$ est utilisée. En reconnaissance d'écriture, la fonction de coût de classification temporelle connexioniste (CTC) [Gra+06] est généralement utilisée lors de la phase d'entraînement. Lors de la phase d'inférence, un décodage glouton de la séquence de probabilité de caractères $\hat{\boldsymbol{y}}$ (*best path decoding*), ou un décodage par recherche en faisceau (*beam search*), est à la place utilisé. La distance de Levenshtein [Lev+66], aussi appelée distance d'édition, permet ensuite de mesurer les taux d'erreur caractère et mot, qui seront introduits dans la section 1.4.

On cherche alors à trouver une fonction de *mapping* optimale F^* minimisant la fonction de coût $\mathcal{L}$ sur les données d'entraînement fournies $\mathcal{D}$:

$$F^* = \arg\min_{F} \sum_{(\boldsymbol{X},\boldsymbol{y})\in\mathcal{D}} \mathcal{L}(\mathbf{y}, F(\boldsymbol{X})) \tag{1.2}$$

Cependant, trouver la fonction *mapping* optimale F^* étant un problème extrêmement complexe, l'objectif de la reconnaissance d'écriture manuscrite est alors de trouver une bonne approximation $\hat{F}$ sur l'ensemble des données d'entraînements $\mathcal{D}$. Nous verrons qu'il s'agit ici d'un problème d'optimisation non convexe pour lequel on trouve un optimum local par descente de gradient.

1.3 Notion de séquence à séquence

Le problème de reconnaissance d'écriture manuscrite, et plus spécifiquement dans le cas de lignes de texte, peut être vu comme un problème de séquence à séquence (*seq2seq*). Le texte présent dans les images à reconnaître possède en effet un ordre de lecture séquentiel,

avec par exemple pour des lignes de texte en langues latines, un ordre de lecture de la gauche vers la droite.

Les approches existantes utilisent ainsi cette connaissance et décomposent l'image $\boldsymbol{X}$ en une séquence de trames T de longueur L. Chaque trame est composée d'une ou plusieurs colonnes de pixels, avec potentiellement du chevauchement d'une trame à une autre. Une illustration de la décomposition en trames est présentée à la figure 1.1.

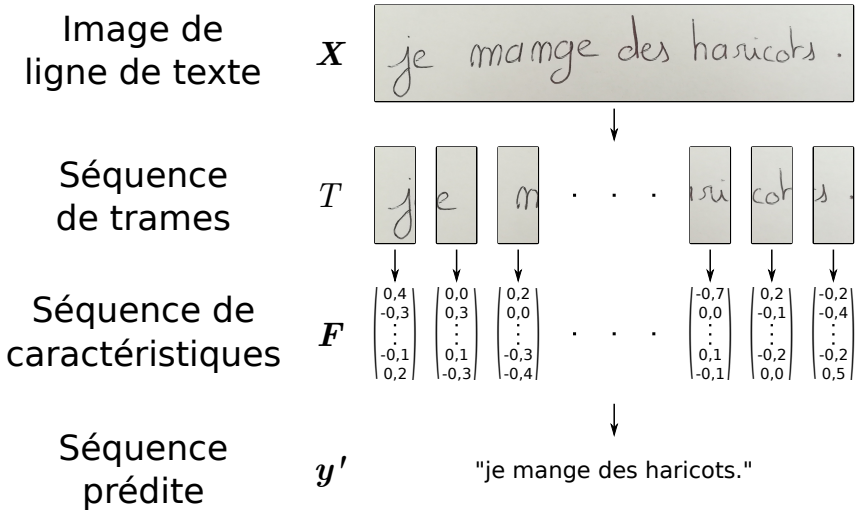


FIGURE 1.1 – Illustration de la décomposition en trames de l'image ainsi que de la séquence de caractéristiques correspondante, et de la prédiction produite en inférence. Dans cet exemple, la taille des trames a été volontairement exagérée. En pratique, les trames sont plus petites que la taille des caractères. Il y a donc plus de trames que de caractères dans la séquence à prédire, ainsi que du chevauchement entre les trames.

Cette décomposition en trames permet en particulier d'organiser l'image de ligne de texte sous la forme d'une séquence, et donc de traiter l'image de manière séquentielle pour identifier les différents caractères présents.

Les approches de reconnaissance de lignes d'écriture manuscrite calculent une (ou une suite de) séquence de caractéristiques, $\boldsymbol{F} \in \mathbb{R}^{L \times D}$, qui représente l'information contenue dans l'image, avec D le nombre de caractéristiques. La prédiction $\hat{y} \in \Sigma^{L'}$ est alors obtenue sous la forme d'une séquence de longueur L'. Selon la fonction de coût utilisée, L' est potentiellement différent du nombre de trames L.

1.4 Métriques d'évaluation CER et WER

Pour évaluer les performances d'un modèle de reconnaissance d'écriture, le taux d'erreur caractères (CER) et taux d'erreur mots (WER) sont le plus souvent utilisés. Le

CER (respectivement WER) est basé sur la distance d'édition [Lev+66], qui détermine le nombre de changements minimaux au niveau des caractères (respectivement mots) ; c'est-à-dire le nombre d'insertions, de suppressions ou de substitutions de caractères (respectivement mots) ; entre deux séquences $\boldsymbol{x}$ et $\boldsymbol{y}$. En dénotant i, le nombre d'insertion, d, le nombre de suppressions, et s, le nombre de substitutions entre deux séquences $\boldsymbol{x}$ et $\boldsymbol{y}$, la distance d'édition ED est alors définie de la manière suivante :

$$ED(\boldsymbol{x}, \boldsymbol{y}) = i + d + s \tag{1.3}$$

Pour calculer cette distance d'édition, l'algorithme de programmation dynamique Wagner-Fischer est généralement utilisé [WF74].

Enfin, le CER (respectivement WER) est obtenu en normalisant par le nombre d'éléments dans la séquence vérité terrain $\boldsymbol{y}$, c'est-à-dire le nombre de caractères (respectivement mots) dans $\boldsymbol{y}$:

$$CER(\boldsymbol{x}, \boldsymbol{y}) = \frac{ED(\boldsymbol{x}, \boldsymbol{y})}{|\boldsymbol{y}|} = \frac{i + d + s}{|\boldsymbol{y}|} \tag{1.4}$$

Première partie

État de l'art

L'apprentissage automatique est un procédé qui consiste à développer des modèles d'intelligence artificielle capables d'apprendre à résoudre des tâches à partir de données qui leur sont présentées. L'apprentissage automatique se révèle très utile pour résoudre des problèmes particulièrement complexes pour des programmes ou algorithmes traditionnels. Par exemple, l'apprentissage automatique est utilisé pour résoudre des problèmes comme la traduction d'une langue à une autre, la classification d'images, la conduite de voitures autonomes ou encore la reconnaissance d'écriture manuscrite.

Les réseaux de neurones artificiels (ou réseaux de neurones) ont été proposés pour la première fois dans les années 1950. Ces modèles sont basés sur les réseaux de neurones biologiques et cherchent à reproduire leurs comportements et leurs capacités d'apprentissage. Ils ont récemment gagné en popularité, accompagnés de nombreuses innovations. Les réseaux de neurones ont permis d'améliorer les résultats existants dans de nombreux domaines et demeurent aujourd'hui la méthode la plus fréquemment utilisée, offrant les meilleurs résultats.

Dans le domaine de la reconnaissance d'écriture, les réseaux de neurones ont progressivement remplacé les approches basées sur des modèles de Markov cachés. Les réseaux de neurones offrent de nombreux avantages comme par exemple : de meilleures capacités de modélisation contextuelles ; la disparition de la contrainte de segmentation en caractères ; ainsi que l'apprentissage automatique de caractéristiques complexes.

Ces modèles utilisent les données qui leurs sont fournies pour s'entraîner, et cherchent à obtenir les meilleurs résultats possibles sur les données d'entraînement. Dans le cadre d'apprentissage supervisé, les données fournies nécessitent d'être annotées au préalable afin de fournir à ces modèles des exemples accompagnés des réponses attendues.

Aujourd'hui, la tendance dans le domaine de la reconnaissance d'écriture manuscrite, ainsi que dans bien d'autres domaines, est de proposer des réseaux de neurones bien plus larges, profonds et complexes pour obtenir de meilleurs résultats. Cependant cette augmentation de la complexité des modèles demande en contrepartie bien plus de données annotées. En revanche, il est à noter qu'utiliser un modèle plus large avec une même quantité de données n'améliore pas forcément les résultats. Avec peu de données et un surnombre de poids entraînables, un modèle aura alors tendance à se reposer sur des caractéristiques spécifiques aux données présentées au modèle. Celui-ci ne parvient alors pas à généraliser sur de nouvelles données. On parle alors de surapprentissage.

Il est donc nécessaire de disposer de suffisamment de données annotées pour entraîner convenablement ces modèles de plus en plus larges. Cependant, ces données peuvent être particulièrement coûteuses à obtenir, comme dans le domaine de la reconnaissance

d'écriture manuscrite ancienne où les écritures sont complexes à reconnaître et sujettes à de nombreuses dégradations.

Dans cette première partie, nous étudierons l'état de l'art des approches de réseaux de neurones pour la reconnaissance d'écriture manuscrite. Le chapitre 2 se concentrera en particulier sur les architectures habituellement utilisées dans le domaine. Le chapitre 3 introduira l'architecture Transformer [Vas+17] dans le contexte du traitement automatique de la langue. La majorité des travaux de la thèse étant basée sur ces approches, nous détaillerons le fonctionnement de principes clés, tels que le mécanisme d'attention à têtes multiples. Par la suite, le chapitre 4 présentera comment les couches de Transformer ont pu être intégrées dans les approches les plus récentes de reconnaissance d'écriture manuscrite. Enfin, le chapitre 5 introduira en détail les problématiques liées aux documents anciens et au manque de données.

Chapitre 2

Réseaux de neurones pour la reconnaissance d'écriture manuscrite

Dans ce chapitre, nous étudions les différentes architectures utilisées dans le domaine de la reconnaissance d'écriture manuscrite ces dernières années. Nous porterons un accent particulier sur les avantages et inconvénients de ces architectures.

Dans un premier temps, la section 2.1 présentera les jeux de données modernes les plus utilisés dans le domaine de la reconnaissance d'écriture manuscrite. La section 2.2 étudiera ensuite les différentes architectures basées sur l'utilisation de couches récurrentes, comme par exemple les architectures CRNN (également appelées CNN-LSTM). Dans la section 2.3, nous étudierons les architectures basées sur le paradigme encodeur-décodeur et sur l'utilisation de mécanismes d'attention. La section 2.4 étudiera les architectures totalement convolutives qui ont été proposées pour remédier aux défauts liés aux architectures précédentes, basées sur l'utilisation de couches récurrentes. Enfin, dans la section 2.5, nous ferons un bilan du chapitre, et des différents avantages et inconvénients.

2.1 Jeu de données de reconnaissance d'écriture modernes

Cette section présente les jeux de données de reconnaissance d'écriture manuscrite moderne IAM et RIMES. Ils seront présentés dans les sections 2.1.1 et 2.1.2 respectivement.

Ces jeux de données modernes, et en particulier IAM, sont les plus utilisés par les différentes approches pour se comparer. Dans le but de valider nos approches, nous évaluerons nos approches sur ces deux jeux de données et nous nous comparerons à l'état de l'art dans le chapitre 10.

2.1.1 IAM

La base de données de reconnaissance d'écriture manuscrite hors-ligne IAM [MB02] est la base de données la plus utilisée par les différentes approches de l'état de l'art pour se comparer.

Pour obtenir ce jeu de données, 657 scripteurs ont rédigé dans une écriture anglaise moderne des paragraphes provenant du corpus de texte anglais Lancaster - Oslo/Bergen (LOB) [SLG78]. Un total de 1 539 pages a alors été obtenu. De plus, la base fournit ces pages dans un format de lignes isolées ainsi que de mots isolés, pour un total de 13 353 lignes et 115 320 mots annotés. Au cours de cette étude, nous nous concentrons sur les images de lignes de texte isolées, ainsi que sur leur annotation. Des exemples de lignes de textes sont disponibles dans la figure 2.1.

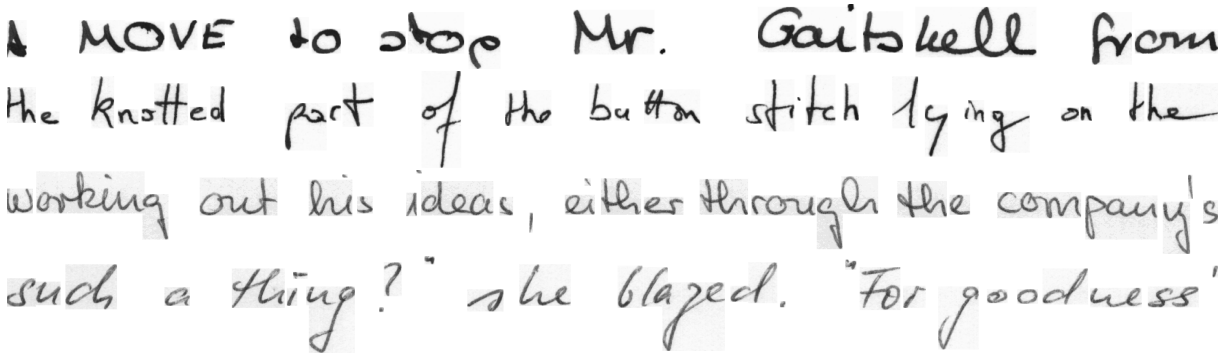


FIGURE 2.1 – Exemple de lignes de texte annotées provenant du jeu de données IAM [MB02].

Les auteurs de la base de données ont par ailleurs mis à disposition des chercheurs, des ensembles d'entraînement, de validation et de test pour évaluer et comparer les différentes approches. Il en résulte alors : 6 161 lignes isolées pour l'ensemble d'entraînement ; deux ensembles de validation composés de 900 et 940 lignes ; ainsi qu'un ensemble de test de 1 861 lignes. Dans ce partitionnement, l'écriture de chaque auteur ayant contribué n'apparaît que dans un seul ensemble.

Cependant, le partitionnement "aachen" est beaucoup plus communément utilisé par la communauté de la reconnaissance d'écriture pour entraîner des systèmes de reconnaissance d'écriture et se comparer à d'autres approches. Ce second partitionnement contient alors 6 482 lignes pour l'ensemble d'entraînement, 976 pour la validation, et 2 915 pour l'ensemble de test. C'est par ailleurs le partitionnement que nous utilisons dans les différentes expériences présentées.

Ce jeu de données est l'un des plus majeurs dans le domaine. Il contient un nombre important de lignes annotées, ce qui en fait un avantage notable. Cependant, ce nombre de données peut se révéler être insuffisant pour entraîner des architectures profondes.

2.1.2 RIMES

Le jeu de données RIMES [Aug+06] propose de nombreux documents manuscrits modernes rédigés en français.

Il est composé de divers documents obtenus à partir de scénarios fictifs dans lequel il a été demandé à un scripteur de rédiger un document concernant une demande administrative. Ces documents sont généralement sous la forme de lettres, ou de fac-similés (fax). Au total, plus de 1 300 scripteurs ont rédigé 5 605 messages pour un total de 12 723 pages.

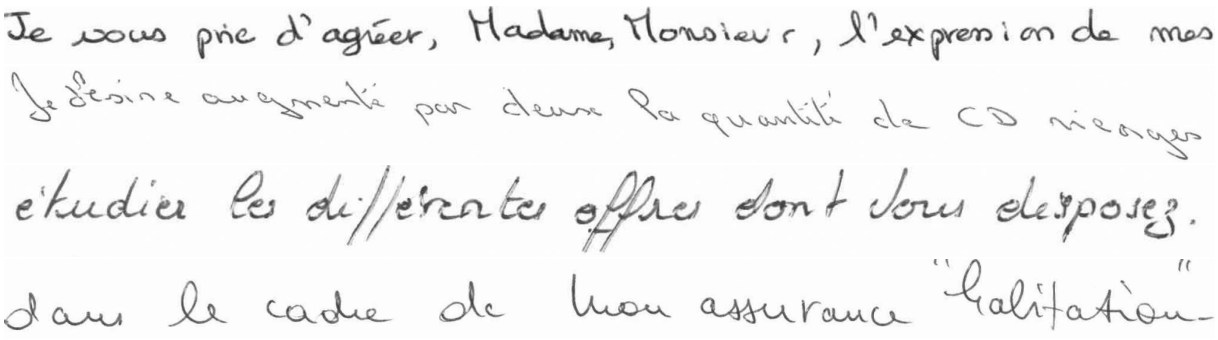


FIGURE 2.2 – Quelques lignes de texte manuscrit provenant du jeu de données RIMES [Aug+06].

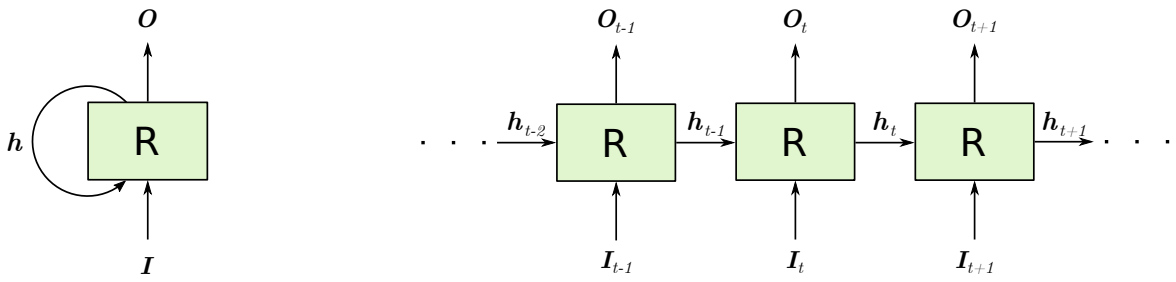
Dans notre cas, nous nous intéressons à la tâche de reconnaissance de lignes de textes isolées. Pour cela, nous considérons le sous-ensemble de données qui ont été segmentées et annotées au niveau des lignes lors de la compétition de l'ICDAR 2011 [GE11]. Des exemples de lignes issues du jeu de données sont disponibles dans la figure 2.2. Dans notre cas, nous utilisons un partitionnement fréquemment utilisé, et comportant 9 947 lignes en apprentissage, 1 333 lignes en validation, et 778 lignes pour le jeu de test.

2.2 Architectures récurrentes traditionnelles

2.2.1 Couches récurrentes

Les réseaux de neurones sont devenus la solution la plus utilisée pour résoudre des problèmes de reconnaissance d'écriture manuscrite. Les premières architectures du domaine

à avoir été popularisées se basaient sur des couches récurrentes [Liw+07 ; Gra+08]. Les couches récurrentes sont très intéressantes car elles sont capables de traiter des séquences de caractéristiques de manière séquentielle, tout en mémorisant de l'information. Elles ont aussi l'avantage de pouvoir traiter des séquences de longueur variable.



(a) Couche récurrente (b) Couche récurrente "dépliée"

FIGURE 2.3 – Schéma d'une couche récurrente simple. La figure (2.3b) illustre en particulier la forme "dépliée" permettant de mieux visualiser le calcul séquentiel réalisé par des couches récurrentes.

Une couche récurrente simple R est composée d'un ensemble de neurones traitant une séquence de caractéristiques de longueur L : $\boldsymbol{I} = (\boldsymbol{I}_1, \boldsymbol{I}_2, ..., \boldsymbol{I}_L)$. Une couche récurrente procède à un calcul séquentiel de la séquence $\boldsymbol{I}$, traitant lors de l'étape t, le vecteur de caractéristiques $\boldsymbol{I}_t$ correspondant à la trame t de l'image. Pour ce faire, une couche récurrente simple est composée d'un ensemble de neurones avec des connexions classiques à propagation avant, mais aussi des connexions récurrentes. Ces connexions récurrentes permettent d'inclure une information supplémentaire $\boldsymbol{h}_{t-1}$ (sous la forme d'un vecteur), appelée état caché, et résultant de l'activation de R à l'étape précédente $(t-1)$. La sortie $\boldsymbol{O}_t$ de la couche récurrente R est alors obtenue en fonction de l'entrée $\boldsymbol{I}_t$ ainsi que de l'information contextuelle récurrente provenant de l'état caché $\boldsymbol{h}_{t-1}$, comme illustré dans la figure 2.3. On obtient alors la séquence de caractéristiques de sortie de la couche $\boldsymbol{O}$ en concaténant les sorties calculées pour chaque trame de l'image.

L'utilisation de ces couches a permis une amélioration nette du taux de reconnaissance dans le domaine de la reconnaissance d'écriture grâce à l'intégration d'un contexte passé [Liw+07]. Les réseaux récurrents ont par exemple permis d'atteindre un WER de 25,9% [Gra+08] sur la base de données d'écritures manuscrites modernes IAM [MB02].

En revanche, si les couches récurrentes sont en théorie capables d'apprendre des contextes larges, c'est-à-dire d'accumuler de l'information provenant de trames précédentes lointaines, en pratique, ces couches sont vite limitées [HS97 ; Gra+08]. Cette difficulté est due à la rétropropagation du gradient à la fois sur la profondeur du réseau, mais aussi à travers le temps. Les gradients provenant d'étapes lointaines tendent alors

exponentiellement vers une valeur nulle [HS97]. Ce phénomène est fortement connu dans la communauté des réseaux de neurones et se nomme le problème de la disparition du gradient (*vanishing gradient problem* en anglais).

Long Short-Term Memory

Pour contrer le problème de la disparition du gradient dans les réseaux récurrents, Hochreiter et Schmidhuber ont proposé les couches de mémoire court et long terme (ou *Long Short-Term Memory* (LSTM)) [HS97]. Ces couches sont, elles aussi récurrentes, mais elles intègrent une cellule capable de mémoriser de l'information, ainsi qu'une porte d'entrée et une porte de sortie permettant de filtrer l'information entrante ou sortante de la cellule. En plus de l'état caché h, une couche LSTM transmet aussi l'état de la cellule, c, dans le but de fournir une information contextuelle. Cette cellule, ainsi que les portes d'activations permettent ainsi aux couches LSTM d'être bien plus robustes au problème de la disparition du gradient.

Plus tard, Gers et al. [GSC00] ont proposé une amélioration des LSTM, en ajoutant la possibilité d'oublier de l'information retenue avec une porte d'oubli. Ces couches peuvent alors apprendre à oublier les informations qu'elles jugent inutiles, ne retenant que celles qu'elles estiment essentiel.

Cela se révèle particulièrement utile pour la tâche de reconnaissance d'écriture, et l'intégration d'informations contextuelles lointaines. La très grande majorité des approches récurrentes dans le domaine se base ainsi sur des couches LSTM [Liw+07 ; Gra+08 ; GS09 ; Blu+14 ; DZN16 ; SBY16 ; Mic+19].

Réseaux récurrents multi-directionnels

Couches récurrentes bidirectionnelles Les réseaux récurrents se servent du contexte passé pour reconnaître l'écriture manuscrite. Cependant, il peut aussi être utile en reconnaissance d'écriture hors-ligne, d'utiliser de l'information provenant de contextes futurs, dans le sens de l'ordre de lecture. Pour cela, les réseaux récurrents bidirectionnels ont été proposés par Schuster et al. [SP97]. Ils combinent deux réseaux récurrents pour traiter l'information et intégrer le contexte passé et futur. Le réseau récurrent "avant" traite la séquence dans l'ordre de la lecture (de gauche à droite pour des écritures latines), tandis que le réseau "arrière" traite la séquence dans l'ordre inverse à la lecture (de droite à gauche pour les écritures latines). Les résultats des deux réseaux sont ensuite fusionnés, en général au moyen d'une concaténation des vecteurs de sortie des réseaux "avant" et

"arrière". Le vecteur résultant contient alors de l'information d'un contexte passé, mais aussi futur.

Les réseaux récurrents bidirectionnels (BLSTM) basés sur des couches LSTM uniquement, ont été très largement utilisés dans le domaine de la reconnaissance d'écriture manuscrite. Les architectures proposées sont ainsi composées à la majorité de ces couches récurrentes, et montrent de bons résultats [Liw+07 ; Gra+08 ; FU15 ; LWH15 ; RRC15].

Dans le domaine de la reconnaissance d'écriture manuscrite, les BLSTM ont été appliqués, et sont par exemple capables d'obtenir des CER de l'ordre de 10 à 20% sur la base IAM [FU15 ; LWH15].

Couches récurrentes multi-dimensionnelles Tandis que les réseaux récurrents bidirectionnels ont été proposés pour intégrer l'information provenant de deux directions, c'est-à-dire le long de la dimension séquentielle, Graves et al. [GFS07] proposent les réseaux récurrents multi-dimensionnels pour généraliser à un nombre de dimensions quelconques. Dans le domaine de la reconnaissance d'écriture, ils sont utilisés pour intégrer de l'information provenant de contextes en deux dimensions. Ces réseaux intègrent alors de l'information provenant des directions horizontales et verticales. Les réseaux récurrents multi-dimensionnels sont particulièrement utiles pour reconnaître : les parties ascendantes et descendantes des caractères ; des lignes de texte obliques ; des paragraphes ou pages complètes ; ou encore dans les cas où de fortes variabilités d'échelles peuvent être observées. Les couches récurrentes multi-dimensionelles ont été appliquées, notamment pour des langues non-latines comme l'arabe ou le chinois dans lesquelles de petites variations ou accentuations des caractères sont particulièrement importantes [GS09 ; Blu+14 ; Moy+14 ; ML15 ; VDN16]. Cependant, elles possèdent un plus grand nombre de paramètres et demandent en conséquence plus de données annotées.

Limite des couches récurrentes

Cependant, ces architectures souffrent d'un problème inhérent aux couches récurrentes. Ces couches effectuent un calcul séquentiel, trame par trame, et nécessitant le résultat de la trame précédente. Les couches récurrentes ne tirent donc pas profit du parallélisme et de l'accélération offerte par les cartes graphiques modernes [OJ04 ; RMN09]. Les architectures basées sur des couches récurrentes sont alors lentes et coûteuses à entraîner. Ce problème est accentué pour les architectures multi-dimensionnelles qui utilisent plus de couches récurrentes.

2.2.2 Couches de convolution

Dans le domaine de la classification d'images, les réseaux de neurones convolutifs ont connu un très fort intérêt ces dernières années [KSH12 ; SZ14 ; He+16 ; Sze+16 ; Hua+17 ; Xie+17]. Ces réseaux font usage de couches de convolution, qui ont l'avantage d'être invariantes aux translations. Ces couches sont basées sur l'utilisation de filtres qui appliquent un opérateur de convolution sur les différentes parties de l'image. Les filtres utilisés ont généralement une taille assez petite (souvent 3×3, 5×5 ou 7×7), pour permettre l'extraction de caractéristiques locales à l'image. Ces filtres sont appliqués sur des parties consécutives en suivant le principe de fenêtre glissante, tout en gardant les mêmes poids. En particulier, cette opération peut être massivement parallélisée sur les différentes parties de l'image, grâce aux cartes graphiques modernes.

De plus, ces couches de convolution sont combinées à des couches d'activation non-linéaires, telle que la fonction ReLU (pour *Rectified Linear Unit*), pour modéliser des fonctions complexes. D'autre part, des fonctions d'agrégation (ou de *pooling*) sont utilisées pour réduire la dimensionalité des activations intermédiaires.

En conséquence, les couches de convolutions ont de nombreux avantages. Du fait qu'elles utilisent des poids partagés et des filtres de tailles généralement réduites, les couches de convolution sont relativement légères. Cependant, pour modéliser des formes complexes, les couches de convolutions ont besoin d'avoir accès à un champ réceptif (c'est-à-dire la partie de l'image sur laquelle une couche possède de l'information) large. Les réseaux convolutifs utilisent alors une succession de plusieurs couches pour agrandir la taille du champ réceptif [SZ14 ; He+16 ; Sze+16 ; Hua+17 ; Xie+17]. Elles nécessitent en conséquence un jeu de données suffisamment large et varié pour avoir de bonnes capacités de généralisation. Enfin, un avantage majeur de ces couches réside dans le fait quelles apprennent à fournir automatiquement des caractéristiques pertinentes pour la tâche considérée. C'est un avantage notable là où auparavant, il était nécessaire de calculer en amont des caractéristiques choisies manuellement pour entraîner des réseaux de neurones.

Grâce aux couches de convolution, de nombreuses avancées et d'excellents résultats ont été permis dans le domaine de la classification d'image [KSH12 ; SZ14 ; He+16 ; Sze+16 ; Hua+17 ; Xie+17].

2.2.3 Architectures convolutives et récurrentes

Les couches de convolutions ont été intégrées dans les architectures récurrentes existantes grâce à leurs avantages [ML15 ; DZN16 ; SBY16 ; SDN16 ; VDN16 ; BM17 ; Pui17 ; WYL17 ; Dut+18 ; Sou+19]. D'une part, elles permettent l'apprentissage automatique de caractéristiques. D'autres part, elles améliorent les performances des réseaux existants grâce à leurs facultés de reconnaissance de formes dans les images.

Architecture CRNN

Les architectures CRNN (pour *Convolutional Recurrent Neural Network*) combinent des couches de convolutions avec des couches récurrentes bidirectionnelles, et sont couramment utilisées dans le domaine de la reconnaissance d'écriture manuscrite [DZN16 ; SBY16 ; SDN16 ; BM17 ; Pui17 ; WYL17 ; Dut+18 ; Sou+19]. Notons qu'elles sont aussi référencées dans la communauté par le terme CNN-LSTM. Une succession de couches de convolutions est utilisée pour extraire des caractéristiques pertinentes sur l'image d'entrée. Elles permettent aussi de réduire au fur et à mesure la dimension des vecteurs (ou cartes de caractéristiques) intermédiaires, grâce à des couches de *pooling*. Une succession de couches BLSTM est ensuite utilisée pour intégrer le contexte séquentiel et reconnaître la séquence de caractères. Les architectures CRNN se sont imposées dans le domaine de la reconnaissance d'écriture manuscrite et permettent d'obtenir des CER descendants jusqu'à 5,47% sur IAM [Pui17 ; Dut+18 ; WZG22].

Architecture convolutives multi-dimensionelles

Messina et al. [ML15] proposent d'intégrer des couches de convolution en combinaison avec des couches récurrentes multi-dimensionelles. Leur architecture se base sur une alternance entre couches récurrentes multi-dimensionnelles et couches de convolution. Ces architectures convolutives multi-dimensionelles peuvent montrer de meilleurs résultats que les approches CRNN, mais sont en général bien plus complexes à entraîner et demandent donc en contrepartie plus de données annotées [ML15 ; VDN16].

2.2.4 Approche de classification temporelle connexioniste

La tâche de reconnaissance d'écriture manuscrite se base traditionnellement sur des architectures de réseaux de neurones utilisant des trames, où à chaque trame est associée une partie de l'image (voir figure 1.1 en p. 23). Pour chacune de ces trames, un vecteur

de caractéristique est calculé, avant d'en déduire un caractère associé, ou un vecteur de probabilités caractère par trame (un treillis de probabilité). Cependant, le nombre de trames ; et donc la taille de la séquence de caractères produite par le réseau ; est plus grand que le nombre de caractères présents dans l'image. Une segmentation de l'image en caractères était alors nécessaire pour assurer la correspondance entre les séquences.

Pour résoudre ce problème, Graves et al. [Gra+06] proposent l'approche de classification temporelle connexioniste (ou CTC pour *Connectionist Temporal Classification*) pour la reconnaissance automatique de la parole, avant de l'appliquer à la reconnaissance d'écriture manuscrite [Liw+07 ; Gra+08].

Cette approche permet de résoudre les problèmes d'alignement entre la séquence de sortie produite par un réseau de neurones et la séquence labellisée, tout en associant un score de proximité entre les deux séquences. Pour ce faire, l'approche détermine la probabilité de la séquence de caractères désirée (la vérité terrain) dans le treillis de probabilité produit par le réseau. Un chemin dans le treillis produit la séquence désirée selon deux critères : la possibilité de produire une suite consécutive de labels identiques pour un même caractère dans la séquence ; ainsi que l'utilisation d'un jeton (ou caractère) spécial "`Blank`" pour indiquer une absence de prédiction associée à une trame. De plus, ce jeton sert aussi à séparer des caractères distincts dans le cas particulier où le réseau doit produire une suite de caractères identiques consécutifs (comme pour le bigramme "`ff`" dans "coffre").

Dans le domaine de la reconnaissance d'écriture manuscrite, l'approche CTC s'est imposée comme la référence pour entraîner des réseaux de neurones.

2.3 Architectures encodeur-décodeur et mécanismes d'attention

2.3.1 Principe des architectures encodeur-décodeur

Récemment, le paradigme d'architectures encodeur-décodeur s'est largement développé pour résoudre des tâches de séquence à séquence. Ces différentes tâches ont la particularité d'utiliser une séquence en entrée de réseau, et de fournir une autre séquence en sortie. C'est par exemple le cas de la traduction automatique, de la reconnaissance de la parole, ou de la reconnaissance de l'écriture.

Les architectures encodeur-décodeur sont basées sur deux parties principales. La partie encodeur traite la séquence d'entrée et cherche à obtenir une représentation compacte de celle-ci, en résumant les informations importantes qui y sont contenues. La partie décodeur utilise quant à elle cette représentation dans le but de produire la séquence de sortie. Au moyen d'un processus autorégressif, le décodeur produit la séquence de sortie étape par étape. Il utilise pour cela à la fois l'information de la représentation compacte, mais aussi les parties déjà produites par le décodeur. Ces architectures sont généralement entraînées à partir d'une fonction de coût basée sur l'entropie croisée (cross-entropy).

2.3.2 Mécanismes d'attention

Principe et intérêts des mécanismes d'attention

Les mécanismes d'attention ont pour fonction de filtrer des parties de l'information en donnant une forte importance aux parties jugées intéressantes, tout en réduisant l'importance des zones jugées non utiles. L'avantage majeur des mécanismes d'attention réside dans le fait qu'ils peuvent être appris automatiquement grâce aux algorithmes d'apprentissages. Différents mécanismes d'attention séquentiels peuvent être utilisés comme l'attention de Bahdanau [BCB14] ou l'attention à base de requêtes, clés et valeurs (query, key et value) typique des architectures Transformer [Vas+17]. Des mécanismes d'attention peuvent aussi être utilisés pour filtrer des caractéristiques au sein même de couches au moyen de mécanismes de portes [HS97 ; Bra+16 ; BM17 ; YHM20].

Ces mécanismes d'attention sont très utiles au sein des architectures encodeur-décodeur puisqu'ils permettent au décodeur de porter son attention sur des parties différentes de la représentation produite par l'encodeur. Cette attention évolue au cours du temps pour prédire les différents caractères successifs. De plus, les mécanismes d'attention sont très utiles puisqu'ils permettent d'expliquer un modèle en visualisant sur quelles parties de l'image un modèle porte son attention.

Intégration des mécanismes d'attention dans des architectures récurrentes

Doetsch et al. [DZN16] proposent d'intégrer un mécanisme d'attention dans une architecture de reconnaissance d'écriture. Pour cela, ils utilisent une architecture CRNN en tant que partie encodeur de leur réseau pour résumer la séquence d'entrée. À la suite, un mécanisme d'attention basé sur les travaux de Bahdanau et al. [BCB14] est utilisé et permet de filtrer l'information produite par l'architecture CRNN. Par la suite, un déco-

deur basé sur une couche de LSTM est utilisé pour produire la séquence de sortie sur la base de la sortie de l'encodeur pondérée par la carte d'attention produite. Michael et al [Mic+19] reprennent le principe de l'architecture proposée par Doetsch et al. et comparent différentes variantes liées à l'architecture et au mécanisme d'attention. Ils proposent notamment l'utilisation d'une fonction de coût hybride utilisant les sorties de la partie encodeur, ainsi que celles de la partie décodeur, et observent une amélioration des résultats.

Bluche et al. [BM17] utilisent une architecture plus complexe à base de réseaux récurrents multi-dimensionnels, combinés à un mécanisme d'attention et, d'un décodeur lui aussi basé sur des couches récurrentes multi-dimensionnelles. Grâce à l'utilisation combinée de couches multi-dimensionnelles et d'un mécanisme d'attention, leur architecture est ainsi capable de reconnaître des paragraphes d'écriture et d'apprendre d'elle-même à revenir à la ligne. Cette architecture montre des résultats prometteurs pour la reconnaissance de page mais possède cependant de nombreux poids entraînables. Elle est ainsi complexe à entraîner et demande une grande quantité de données annotées. De plus, son utilisation massive de couches récurrentes en fait une architecture très lente à entraîner et à produire un résultat. Dans un autre article, plutôt que d'utiliser un mécanisme d'attention entre la partie encodeur et la partie décodeur, Bluche et al. [BM17] proposent d'améliorer une architecture CRNN en ajoutant un mécanisme de porte [Bra+16] aux couches de convolution. Ce mécanisme de porte agit de manière similaire à un mécanisme d'attention et permet ainsi de filtrer l'importance des différentes caractéristiques produites.

Les approches basées sur le paradigme d'encodeur-décodeur montrent des résultats intéressants. L'approche proposée par Michael et al. [Mic+19] obtient par exemple un CER de 5,24% sur le jeu de données IAM. L'ajout de mécanismes d'attention permet d'améliorer les résultats et de visualiser l'attention, ce qui rend explicables ces modèles. Cependant, ces approches sont majoritairement basées sur des couches récurrentes, ce qui les rend particulièrement coûteuses à entraîner.

2.4 Réseaux totalement convolutifs

Dans le but de réduire les coûts d'apprentissage, des architectures sans couches récurrentes ont été récemment proposées en reconnaissance d'écriture. Ces architectures sont composées majoritairement de couches de convolution, et sont donc appelées réseaux totalement convolutifs (FCN pour *Fully Convolutional Network*).

Ces réseaux totalement convolutifs améliorent les résultats des approches récurrentes dans le domaine de la reconnaissance d'écriture de textes modernes [Ing+19 ; YHM20 ; CCP21 ; CCP22], mais aussi de textes anciens [YHM20]. Par exemple, sur la base de données de reconnaissance de texte manuscrit moderne IAM, ces approches obtiennent un CER de $4,9\%$ [YHM20 ; CCP22].

2.4.1 Principes des réseaux totalement convolutifs

Les architectures totalement convolutives se distinguent des capacités d'intégration du contexte séquentiel sur l'image de texte, offertes par les couches récurrentes. À la place, ces architectures intègrent des informations contextuelles grâce à un empilement de nombreuses couches de convolutions [Ing+19 ; YHM20 ; CCP21 ; CCP22] et à un champ réceptif large [CCP22]. L'intégration d'informations contextuelles s'effectue alors le long de la profondeur des couches de convolutions [Ing+19], et permet de rivaliser avec des couches récurrentes.

Additionnellement, les architectures totalement convolutives intègrent des mécanismes d'attention à base de portes. Ingle et al. [Ing+19] utilisent ainsi des couches de convolution faisant usage d'un mécanisme d'attention complexe [WH17] en remplacement du mécanisme de porte des LSTM [HS97 ; GSC00]. De même, Yousef et al. [YHM20] proposent des couches de convolution utilisant un mécanisme de porte différent. Ce mécanisme de porte utilise le principe des "réseaux autoroutes" (*highway network*) [SGS15], tirant à la fois profit des connexions résiduelles [He+16], ainsi que des mécanismes de portes des LSTM.

2.4.2 Avantages des réseaux totalement convolutifs

Les architectures totalement convolutives profitent de fortes capacités de parallélisation comparé aux couches récurrentes. Comparées à des architectures récurrentes, elles peuvent donc à la fois s'entraîner et prédire des résultats bien plus rapidement.

Entraînement des architectures totalement convolutives Néanmoins, elle sont complexes tant dans leur schéma que dans leur entraînement [Coq+19]. Elles dépendent de nombreuses couches diverses, utilisant par exemple des mécanismes d'attention et des couches de normalisation. De plus, l'intégration d'information contextuelles globales au travers du calcul de caractéristiques dans les couches de convolution semble complexe. Ces

architectures peuvent alors demander des données annotées variées et en quantité pour être entraînés efficacement. Les architectures proposées par Ingle et al. [Ing+19] ainsi que Yousef et al. [YHM20] utilisent ainsi de très nombreuses augmentations de données afin d'être entraînées convenablement. Ingle et al. [Ing+19] expliquent en particulier que leur architecture totalement convolutive, en comparaison à une architecture récurrente, a besoin de plus de données annotées pour s'entraîner. Grâce à l'utilisation de données privées, leur architecture obtient un CER de 4,0%, et dépasse les résultats des architectures traditionnelles.

Reconnaissance de pages complètes Les architectures totalement convolutives peuvent être utilisées pour reconnaître des paragraphes ou pages complètes. Yousef et al. [YB20] combinent ainsi leur réseau totalement convolutif [YHM20] avec un module composé de couches de convolution pour transformer le problème de reconnaissance d'image de paragraphe en deux dimensions à un problème à une dimension. De même, Coquenet et al. [CCP21] utilisent un encodeur totalement convolutif pour apprendre des caractéristiques à un niveau large et réorganisent l'ordre des caractéristiques pour reconnaître le texte dans des images de paragraphe. Dans une autre architecture proposée par les auteurs [CCP22], l'encodeur est combiné avec un mécanisme d'attention et un décodeur basé sur des couches récurrentes. Ces architectures améliorent les résultats des approches au niveau ligne et obtiennent par exemple entre 3 et 5% de CER sur le jeu de données IAM au niveau page [CCP22 ; YB20]. Comparé aux résultats des approches au niveau ligne, les auteurs expliquent ces différences par l'apport d'informations contextuelles bien plus larges disponibles au niveau des paragraphes [YB20 ; CCP22].

2.5 Discussions

Les architectures CRNN se sont imposées dans le domaine de la reconnaissance d'écriture manuscrite. Des architectures basées sur un modèle d'encodeur-décodeur ont par la suite été proposées, et incluent des mécanismes d'attention pour améliorer la modélisation d'information contextuelles et offre une meilleure qualité de reconnaissance. Cependant, ces architectures reposent sur des couches récurrentes qui impliquent de long temps d'apprentissage et d'inférence. Des réseaux totalement convolutifs ont alors été proposés pour profiter bien mieux de la parallélisation offerte par les cartes graphiques modernes. Cependant, si les réseaux totalement convolutifs semblent montrer de bons résultats, ils n'en

demeurent pas moins complexes à entraîner, notamment à cause de la difficulté à intégrer de l'information contextuelle globale.

Les architectures de Transformer [Vas+17], que nous allons aborder dans le chapitre 3, sont un remplacement efficace au couches récurrentes. Elles permettent en particulier une bien meilleure modélisation d'informations contextuelles et pourraient permettre d'améliorer les architectures existantes.

Chapitre 3

Transformers pour le traitement automatique de la langue

Le traitement automatique de la langue regroupe de nombreuses tâches de type séquence à séquence (modélisation de la langue, traduction automatique, *etc.*), sur lesquelles les approches récurrentes ont été largement prédominantes [BCB14]. Cependant, tout comme pour la reconnaissance d'écriture manuscrite, ces approches sont limitées par la vitesse d'apprentissage des réseaux récurrents.

Pour contrer ce problème, Vaswani et al. [Vas+17] ont proposé l'architecture Transformer. Celle-ci se base sur un mécanisme d'attention particulier : l'attention à têtes multiples. L'attention à têtes multiples permet une analyse sur l'ensemble de la séquence, c'est-à-dire à la fois localement et globalement, sans contrainte de séquentialité dans les calculs. Pour les tâches du traitement automatique de la langue, l'architecture Transformer se révèle efficace pour intégrer de l'information contextuelle [Vas+17 ; Dev+18] et profite du parallélisme offert par les cartes graphiques modernes. Cela en fait donc une excellente alternative aux couches récurrentes, et elle a su s'imposer comme architecture dominante dans les différentes tâches du traitement automatique de la langue [Dev+18].

Dans le domaine de la reconnaissance d'écriture manuscrite, les couches composant l'architecture Transformer pourraient être intégrées dans les architectures existantes. Cela pourrait en particulier offrir de meilleures capacités de modélisation du contexte aux architectures existantes, ainsi que de meilleures vitesses d'entraînement.

La majorité des travaux présentés dans ce manuscrit étant basée sur les architectures Transformer ainsi que sur le mécanisme d'attention à têtes multiples proposés par Vaswani et al. [Vas+17], ce chapitre aborde ces architectures en détail. Dans la section 3.1, nous commencerons par présenter une vue globale de l'architecture Transformer, ainsi que divers principes liés à l'architecture. La section 3.2 abordera quant à elle de manière détaillée le principe clé de l'attention à têtes multiples, puis nous présenterons dans la section 3.3 le détail des blocs d'encodeur et de décodeur. Enfin, la section 3.4 présentera

différents avantages et inconvénients liés à ces blocs, et nous conclurons ce chapitre par une courte discussion dans la section 3.5.

3.1 Architectures Transformer

Dans cette section, nous présentons l'architecture Transformer proposée par Vaswani et al. [Vas+17]. La section 3.1.1 commencera par présenter une vue globale de l'architecture, tandis que nous aborderons en détail les différents principes liés à l'architecture dans les sections suivantes.

3.1.1 Vue globale

Une architecture classique de Transformer est en général basée sur l'utilisation d'un encodeur ainsi que d'un décodeur, composés de blocs de Transformer [Vas+17]. Cependant, selon la tâche considérée, certaines architectures utilisent uniquement un encodeur ou un décodeur. Par exemple, pour la tâche de modélisation de langue, Devlin et al. [Dev+18] utilisent uniquement des blocs d'encodeur Transformer.

Ici, nous proposons une vue globale d'une architecture Transformer typique, basée sur l'utilisation d'un encodeur et d'un décodeur. C'est par exemple le cas pour la tâche de traduction automatique que nous prendrons en exemple. Un schéma global d'une telle architecture est illustré dans la figure 3.1.

L'encodeur et le décodeur utilisent tous deux des séquences différentes en entrée. Ces séquences doivent tout d'abord être converties en une représentation vectorielle au moyen d'un plongement lexical, que nous aborderons dans la section 3.1.2. Par la suite, un encodage positionnel de la séquence est réalisé. Nous détaillerons ce point dans la section 3.1.3.

À partir de la séquence d'entrée qui lui est associée, l'encodeur a pour objectif de calculer et générer une représentation matricielle (sous la forme d'une séquence de vecteurs) riche. Le décodeur utilise cette représentation riche pour produire un résultat final. De plus, le décodeur utilise une autre séquence en entrée, qui correspond à la sortie produite à l'étape précédente par le décodeur. Les blocs d'encodeur et de décodeur Transformer se composent d'un sous-bloc de neurones totalement connectés par position que nous détaillerons dans la section 3.1.4, ainsi que d'attention à têtes multiples qui sera abordée plus tard dans la section 3.2. Les détails des blocs d'encodeur et de décodeurs seront quant à eux expliqués plus tard en section 3.3, une fois les notions importantes abordées.

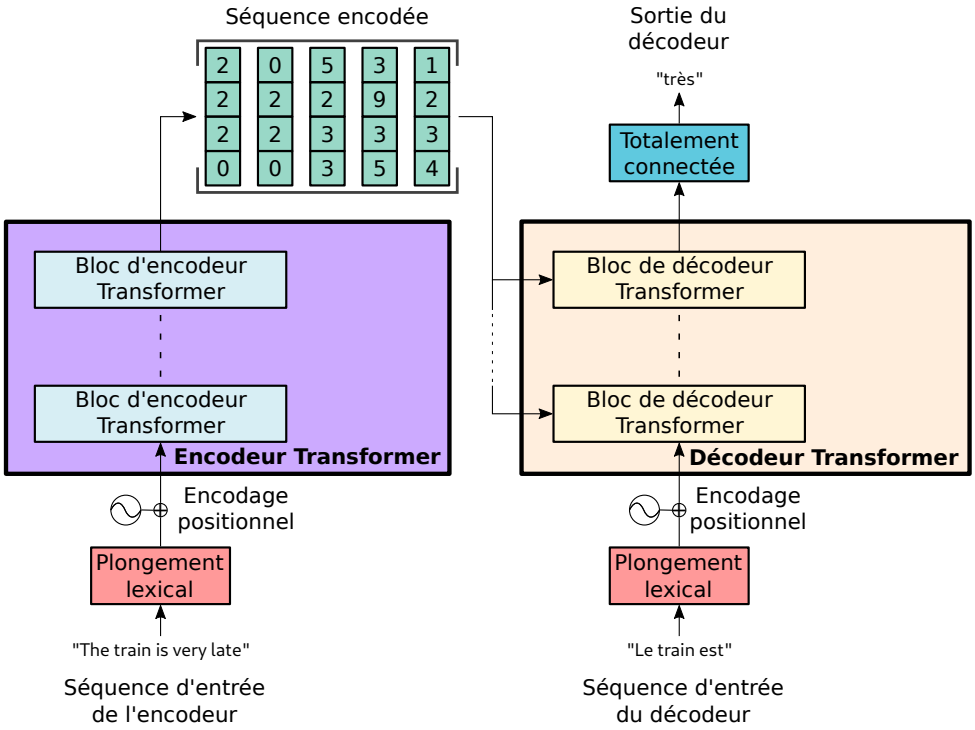


FIGURE 3.1 – Schéma global d'une architecture Transformer composée d'un encodeur ainsi que d'un décodeur, ici, pour la tâche de traduction automatique de l'anglais au français.

Par exemple, pour une tâche de traduction automatique de l'anglais vers le français, l'entrée de l'encodeur correspond à la séquence de texte à traduire. L'encodeur calcule une représentation matricielle de cette séquence riche en information. Le décodeur se sert de cette représentation, ainsi que de la séquence des mots déjà traduits en français pour déterminer quel est le mot français suivant à produire. Ce mot est ensuite ajouté à la séquence des mots déjà traduits pour produire la sortie suivante.

3.1.2 Représentation vectorielle

Les architectures de Transformer permettent de traiter des séquences de vecteur. Quand la séquence n'est pas vectorielle, par exemple pour une séquence de mots ou de caractères, il est nécessaire de la convertir en une représentation vectorielle. En général, un encodage *one-hot* est utilisé pour représenter les différentes valeurs possibles de la séquence. Celui-ci considère un tableau nul avec autant de cases que d'éléments possibles (le nombre de mots ou de caractères), avec une valeur fixée à 1 dans la case correspondant à l'élément. Puis, une couche de neurones totalement connectés est utilisée pour encoder

une représentation plus explicite, et de plus petite dimension. Dans le cas des mots, cet encodage est aussi appelé un plongement lexical (ou *word embedding*).

3.1.3 Encodage positionnel

Le mécanisme d'attention utilisé par les couches d'attention à têtes multiples est une très bonne alternative aux réseaux récurrents de par sa capacité à capturer des dépendances contextuelles. Cependant, ce mécanisme à lui seul ne possède pas de connaissances positionnelles sur les séquences manipulées. Il n'est donc pas capable de faire la différence entre deux mots identiques placés au début ou à la fin de la séquence, ni de savoir quel mot en suit un autre.

Pour contrer cela, une information d'encodage positionnel (sous la forme d'une matrice) est ajoutée à la représentation vectorielle de la séquence grâce à un opérateur d'addition, comme illustré dans la figure 3.1. Le vecteur résultant contient alors à la fois de l'information sur la séquence, ainsi que de l'information positionnelle.

Vaswani et al [Vas+17] proposent un encodage positionnel utilisant des fonctions sinusoïdales pour représenter cette information. L'encodage positionnel sinusoïdal est défini de la manière suivante :

$$\mathrm{PE}(p,i) = \begin{cases} sin\left(\dfrac{p}{10000^{\frac{2k}{D}}}\right) & \text{si } \exists k \in \mathbb{N}, i = 2k \\ cos\left(\dfrac{p}{10000^{\frac{2k}{D}}}\right) & \text{si } \exists k \in \mathbb{N}, i = 2k+1 \end{cases} \tag{3.1}$$

avec p la position dans la séquence, $i \in [\![1; D]\!]$ désignant une caractéristique précise, et D le nombre de neurones utilisé dans l'architecture Transformer. Une visualisation de l'encodage positionnel sinusoïdal est disponible dans la figure 3.2.

La première moitié des caractéristiques de cet encodage positionnel sinusoïdal semble contenir la majorité des informations sur la position. L'autre moitié contient quant à elle peu de variations, laissant plus de place aux informations de la représentation vectorielle. De plus, il est assez facile de prédire l'évolution et le prolongement de cette fonction (figure 3.2), ce qui, selon les auteurs [Vas+17] permettrait de généraliser sur des séquences plus longues que celles vues lors de l'entraînement.

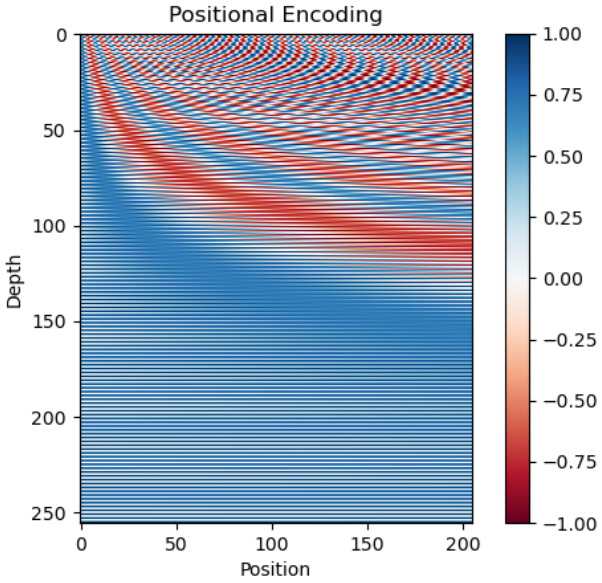


FIGURE 3.2 – Valeurs générées par un encodage positionnel sinusoïdal avec $D = 256$. On peut remarquer que la majorité de l'information positionnelle se localise sur la moitié haute de la matrice. Certains motifs particuliers peuvent être observés, dont la prolongation pour des positions plus lointaines peut être devinée.

3.1.4 Couches de neurones totalement connectés par position

Enfin, les couches de neurones totalement connectés par position (en anglais, *position-wise feed forward network*) sont importantes pour les architectures Transformer. Ces couches sont généralement placées après les couches d'attention (comme illustré dans la figure 3.5).

Cette couche est basée sur l'utilisation de deux couches de neurones totalement connectés, avec une fonction d'activation ReLU [MHN13] entre ces deux couches. Une illustration est disponible dans la figure 3.3. Cette couche a la particularité de traiter les positions des séquences individuellement, en appliquant les mêmes poids sur chaque position. Additionnellement, la dimension intermédiaire, c'est-à-dire entre les deux couches de neurones totalement connectés, est bien plus grande que la dimension en entrée et en sortie de la couche de neurones totalement connectés par position.

Cette couche sert de multiples intérêts. Tandis que les couches d'attention ont pour but d'agréger de l'information contextuelle, les couches de neurones totalement connectés par position effectuent un traitement des caractéristiques orthogonal, c'est-à-dire indépendant de la position. Elles ont alors comme premier intérêt de fournir des caractéristiques plus

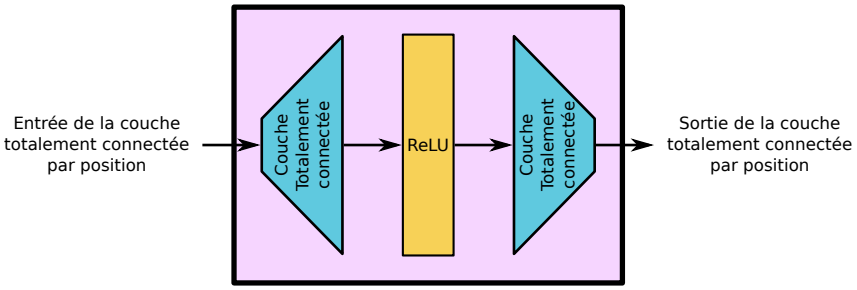


FIGURE 3.3 – Schéma d'une couche de neurones totalement connectés par position. La dimension verticale illustre la taille des caractéristiques intermédiaires.

pertinentes pour chaque position dans la séquence. Pour cela, elles utilisent une dimension intermédiaire plus grande pour produire une représentation plus riche et variée. La deuxième couche totalement connectée agit comme un goulot d'étranglement, et produit une représentation à la fois compacte et pertinente. Cette représentation est alors prête à être utilisée pour des couches d'attention consécutives.

Deuxièmement, elle permet d'ajouter des non-linéarités aux réponses produites par les couches d'attention, grâce à l'utilisation d'une fonction d'activation ReLU. C'est particulièrement utile, car cela offre beaucoup plus d'expressivité au modèle et donc la capacité de s'attaquer à des problèmes complexes.

3.2 Principe de l'attention à têtes multiples

Dans cette section, nous proposons une explication détaillée du fonctionnement sous-jacent du mécanisme d'attention à têtes multiples, qui est une contribution majeure proposée par Vaswani et al. [Vas+17]. Il a été proposé dans le but de fournir une alternative efficace aux couches récurrentes, et il vient généraliser le mécanisme des couches récurrentes.

3.2.1 Mécanisme d'attention

Le mécanisme d'attention des couches de Transformer (aussi appelé *scaled-dot product attention*) est utilisé pour traiter des séquences, dans le but d'intégrer des informations sur l'ensemble du contexte [Vas+17].

Pour illustrer le mécanisme d'attention, nous nous positionnons dans le cas d'une tâche de modélisation de la langue sur une séquence de mots. Pour un mot de cette séquence, le

mécanisme d'attention cherchera à intégrer de l'information sur les mots qui y sont liés. La figure 3.4 illustre ce mécanisme en proposant un exemple.

Pour être traitée par ce mécanisme d'attention, la séquence de mots est convertie en une représentation vectorielle (expliquée en section 3.1.2). Ainsi, on obtient une séquence de vecteurs (ou matrice), $\boldsymbol{S} \in \mathbb{R}^{N \times D}$ composée des différents vecteurs $\boldsymbol{S}_i \; \forall i \in [\![1; N]\!]$.

Le mécanisme d'attention utilise trois séquences de vecteurs en entrée : une séquence de requêtes (Query), $\boldsymbol{Q} \in \mathbb{R}^{N \times D}$; une séquence de clés (Key) $\boldsymbol{K} \in \mathbb{R}^{N \times D}$; et une séquence de valeurs (Value) $\boldsymbol{V} \in \mathbb{R}^{N \times D}$.

Pour chaque vecteur requête $\boldsymbol{Q}_i$ de la séquence de requête $\boldsymbol{Q}$, une requête d'attention est initiée, avec un but propre. Par exemple, si le i-ème mot de la séquence utilisée en entrée correspond à un verbe, la requête pourra avoir pour but de chercher le complément associé à ce verbe. Pour ce faire, la requête $\boldsymbol{Q}_i$ est comparée avec chaque clé $\boldsymbol{K}_j$ de la séquence de clés $\boldsymbol{K}$. Une valeur de corrélation entre $\boldsymbol{Q}_i$ et $\boldsymbol{K}_j$ est calculée au moyen d'un produit scalaire, résultant en un score d'attention :

$$\boldsymbol{A}_{i,j} = \frac{\boldsymbol{Q}_i \cdot \boldsymbol{K}_j}{\sqrt{D}} \tag{3.2}$$

Ainsi, un score d'attention $\boldsymbol{A}_{i,j}$ élevé indiquera une forte corrélation entre la clé $\boldsymbol{K}_j$ et la requête $\boldsymbol{Q}_i$, et donc que la clé $\boldsymbol{K}_j$ répond à la requête $\boldsymbol{Q}_i$. En reprenant l'exemple précédent, on s'attend alors à avoir un score d'attention élevé concernant le complément associé au verbe. Dans le but de limiter le problème de disparition du gradient, les auteurs proposent de remettre à l'échelle ce score d'attention en le multipliant par $\frac{1}{\sqrt{D}}$. Les différents coefficients de score d'attention $\boldsymbol{A}_{i,j}$ peuvent être regroupés dans la matrice (ou séquence de vecteurs) $\mathbf{A} \in \mathbb{R}^{N \times N}$, avec

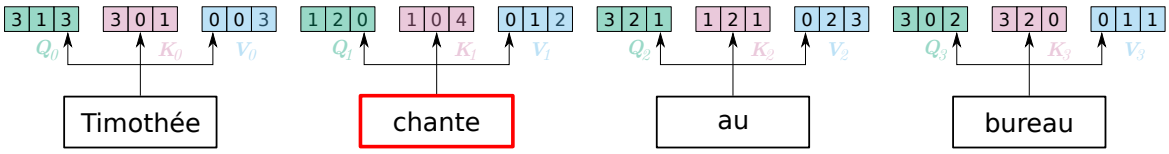
$$\boldsymbol{A} = \frac{\boldsymbol{Q}\boldsymbol{K}^\top}{\sqrt{D}} \tag{3.3}$$

À la suite de cela, une fonction de softmax est utilisée sur le vecteur $\boldsymbol{A}_i$ pour obtenir une séquence de poids d'attention $\boldsymbol{\alpha_i}$, avec comme coefficients :

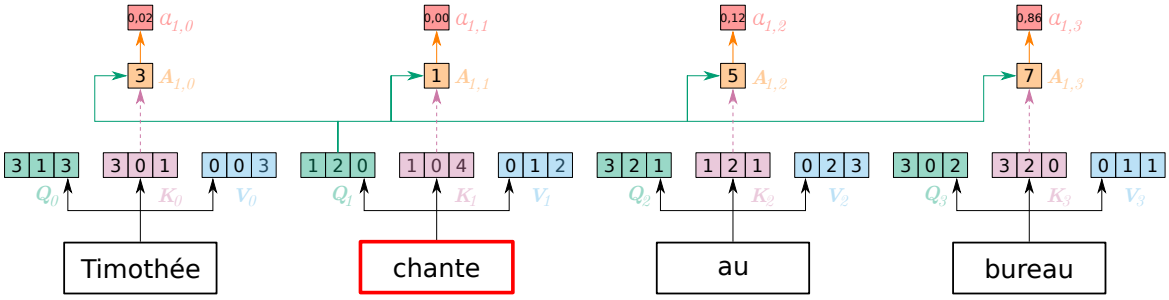
$$\alpha_{i,j} = \frac{e^{\boldsymbol{A}_{i,j}}}{\sum_{j'=1}^{N} e^{\boldsymbol{A}_{i,j'}}} \tag{3.4}$$

Chaque vecteur de valeur $\boldsymbol{V}_j$ est pondéré par son poids d'attention associé $\alpha_{i,j}$:

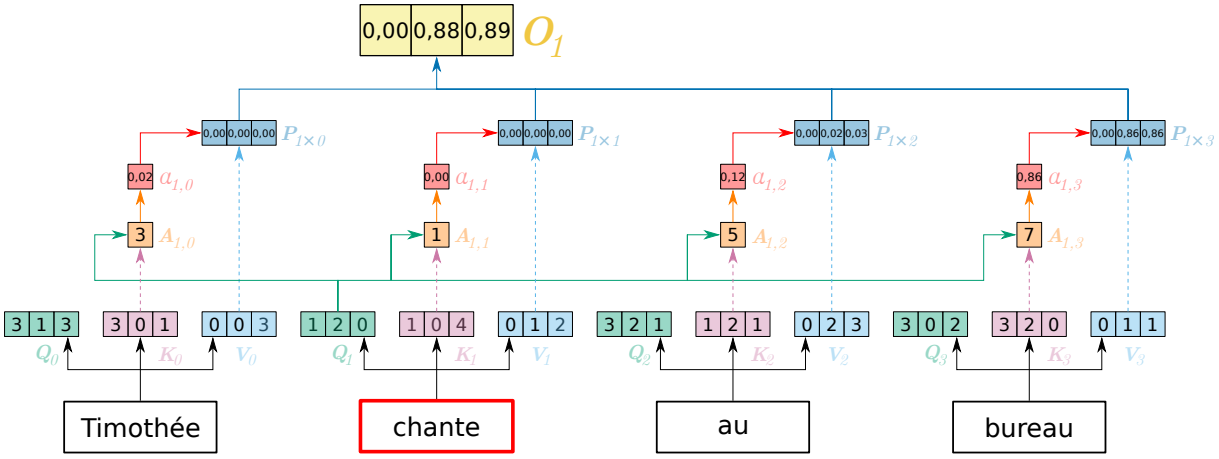
$$\boldsymbol{P}_{i \times j} = \alpha_{i,j} \cdot \boldsymbol{V}_j \tag{3.5}$$



(a) Séquences de vecteurs de requêtes $\boldsymbol{Q}$, de clés $\boldsymbol{K}$ et de valeurs $\boldsymbol{V}$ obtenus à partir d'une séquence de vecteurs, elle-même associée à une séquence de mots. Ces séquences de vecteurs $\boldsymbol{Q}$, $\boldsymbol{K}$ et $\boldsymbol{V}$ sont passées en paramètres d'entrée du mécanisme d'attention.



(b) La requête $\boldsymbol{Q}_1$ est comparée aux différentes clés $\boldsymbol{K}_j$ au moyen d'un produit scalaire, pour obtenir les différents scores d'attention $\boldsymbol{A}_{1,j}$ ainsi que les poids d'attention $\alpha_{1,j}$.



(c) Les poids d'attention $\alpha_{1,j}$ sont utilisés pour pondérer les vecteurs de valeur $\boldsymbol{V}_j$. Les valeurs pondérées $\boldsymbol{P}_{1\times j}$ sont ensuite sommées pour obtenir le résultat du mécanisme d'attention $\boldsymbol{O}_1$ correspondant au vecteur de requête $\boldsymbol{Q}_1$. Il est à noter que $\boldsymbol{O}_1$ ressemble fortement au vecteur de valeur associé au mot "`bureau`" $\boldsymbol{V}_3$. Le mécanisme d'attention trouve donc le complément du verbe dans cet exemple.

FIGURE 3.4 – Illustration du mécanisme d'attention utilisé par les couches de Transformer [Vas+17]. Dans cet exemple, une tâche de traitement automatique de la langue est considérée, avec comme unité lexicale les mots d'une phrase. Nous nous concentrons en particulier sur le vecteur de requête $\boldsymbol{Q}_1$ correspondant au mot "`chante`". Cette requête est construite dans le but de trouver le complément associé à un verbe. Par souci de simplification, la mise à l'échelle par un facteur $\frac{1}{\sqrt{D}}$ est volontairement omise.

$\boldsymbol{P}_{i\times j}$ désigne alors un vecteur de valeurs pondérées. Les valeurs $\boldsymbol{V}$ sont ainsi filtrées sur la base de la corrélation entre les différentes clés $\boldsymbol{K}$ et la requête $\boldsymbol{Q}_i$. Cela permet donc de mettre en valeur certaines parties de la séquence, tandis que d'autres sont masquées, car jugées inintéressantes vis-à-vis de la requête. Dans le cas où le poids d'attention $\alpha_{i,j}$ est élevé, alors le vecteur valeur $\boldsymbol{V}_j$ sera peu différent de $\boldsymbol{P}_{i\times j}$. À l'inverse, une corrélation faible résultera en une valeur fortement filtrée et donc proche du vecteur nul.

Les vecteurs de valeurs pondérées sont sommés pour obtenir le résultat de sortie du mécanisme d'attention correspondant à la requête $\boldsymbol{Q}_i$:

$$\boldsymbol{O}_i = \sum_{j=1}^{N} \boldsymbol{P}_{i\times j} \tag{3.6}$$

On obtient alors la séquence de vecteurs $\boldsymbol{O}$, en mettant bout à bout les vecteurs de sortie $\boldsymbol{O}_i$ correspondant à la réponse à chaque vecteur de requête $\boldsymbol{Q}_i$.

Enfin, en utilisant la notation matricielle, le résultat final peut alors se résumer de la manière suivante :

$$\textsc{Attention}(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}) = \boldsymbol{O} = softmax\left(\frac{\boldsymbol{Q}\boldsymbol{K}^\top}{\sqrt{D}}\right)\boldsymbol{V} \tag{3.7}$$

3.2.2 Attention à têtes multiples

Pour le i-ème mot de la séquence, le mécanisme d'attention est limité à une seule requête, et donc à un seul but. Dans la figure 3.4, le mécanisme d'attention cherche par exemple à trouver quel est le complément du verbe. Or, il serait tout aussi intéressant d'aussi connaître le sujet de ce même verbe. Vaswani et al. [Vas+17] propose alors l'utilisation de plusieurs têtes d'attention pour explorer différents contextes. Ce principe est ainsi communément appelé "attention à têtes multiples". Chaque tête applique ainsi un mécanisme d'attention avec ses propres requêtes $\boldsymbol{Q}$, clés $\boldsymbol{K}$ et valeurs $\boldsymbol{V}$ basées sur la même séquence d'entrée.

Traditionnellement, l'utilisation de plusieurs têtes d'attention est réalisée en utilisant plusieurs représentations ($\boldsymbol{Q}$, $\boldsymbol{K}$ et $\boldsymbol{V}$) de plus petites dimensionnalités. Pour un nombre h de têtes d'attention, la dimension de ces représentations dans chacune de ces têtes est alors égale à $\frac{D}{h}$, dans le but de conserver la dimension totale. Pour la i-ème tête d'attention, une projection est alors réalisée en utilisant des matrices de projection, $\boldsymbol{W}_{\boldsymbol{Q}}^{\boldsymbol{i}}$, $\boldsymbol{W}_{\boldsymbol{K}}^{\boldsymbol{i}}$, et $\boldsymbol{W}_{\boldsymbol{V}}^{\boldsymbol{i}} \in \mathbb{R}^{D\times\frac{D}{h}}$. On a alors :

$$\begin{aligned} \boldsymbol{Q^i} &= \boldsymbol{Q W_Q^i} \\ \boldsymbol{K^i} &= \boldsymbol{K W_K^i} \\ \boldsymbol{V^i} &= \boldsymbol{V W_V^i} \end{aligned} \tag{3.8}$$

avec $\boldsymbol{Q^i}$, $\boldsymbol{K^i}$ et $\boldsymbol{V^i}$ les requêtes, clés et valeurs de la i-ème tête d'attention.

Chaque tête d'attention calcule alors une sortie qui lui est propre, et les sorties sont concaténées les unes avec les autres. En notant $\boldsymbol{O^i}$, le résultat de la i-ème tête d'attention, on obtient alors :

$$\textsc{MultiHead}(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}) = concat\left(\boldsymbol{O^1}, ..., \boldsymbol{O^N}\right) \tag{3.9}$$

3.2.3 Types d'attention

Les couches d'attention utilisées par les architectures Transformer se basent toutes sur le mécanisme d'attention présenté ci-dessus. Cependant, elles se déclinent en différentes variantes selon les séquences qu'elles manipulent ou encore la portée de l'attention.

Objets des mécanismes d'attention

Auto-attention L'auto-attention (ou *self-attention* en anglais), est un mécanisme d'attention dans lequel les vecteurs de requêtes, de clés et de valeurs proviennent d'une même séquence. C'est le cas de l'exemple présenté en figure 3.4. Les vecteurs $\boldsymbol{Q}$, $\boldsymbol{K}$ et $\boldsymbol{V}$ sont ainsi obtenus en passant une même séquence de vecteurs (par exemple, une représentation vectorielle de mots) à trois couches de neurones totalement connectés, produisant chacune un des trois vecteurs. Une requête Q est alors émise sur la base de cette séquence. La requête utilise les clés K de cette séquence et calcule ainsi des poids d'attention pour filtrer les valeurs V de cette même séquence.

Attention mutuelle L'attention mutuelle, ou attention encodeur-décodeur, utilise en revanche deux séquences. Elle est utilisée dans les blocs de décodeur dans lesquels une séquence provient de l'entrée du décodeur (ou du bloc de décodeur précédent), et une autre séquence provient quant à elle de la sortie de l'encodeur. La séquence du décodeur est utilisée pour générer la séquence de requêtes $\boldsymbol{Q}$ en utilisant de même une couche de neurones totalement connectés. La séquence provenant de l'encodeur est quant à elle utilisée pour générer à la fois la séquence de clés $\boldsymbol{K}$ et celles de valeurs $\boldsymbol{V}$. Les requêtes du

décodeur $\boldsymbol{Q}$ sont alors comparées aux clés de l'encodeur $\boldsymbol{K}$ pour calculer une corrélation, avant d'être utilisées pour filtrer les valeurs de l'encodeur $\boldsymbol{V}$.

Causalité du mécanisme d'attention

Si par défaut l'attention permet à une requête d'accéder à l'intégralité de la séquence, certains mécanismes d'attention peuvent nécessiter un principe de causalité.

L'attention causale a pour but de n'autoriser l'accès qu'aux vecteurs de clés précédant une certaine requête $\boldsymbol{Q}_i$. Ainsi, l'ensemble des clés accessibles est défini de la manière suivante : $\mathbf{K}_{\rightarrow i} = \{\boldsymbol{K}_j \mid j \leq i\}$. Pour cela, un masque est utilisé, laissant inchangés les scores d'attention pour les clés autorisées, et fixant le score d'attention des clés non autorisés à $-\infty$. Après application de la fonction $softmax$, les poids d'attention non autorisés sont donc égaux à 0.

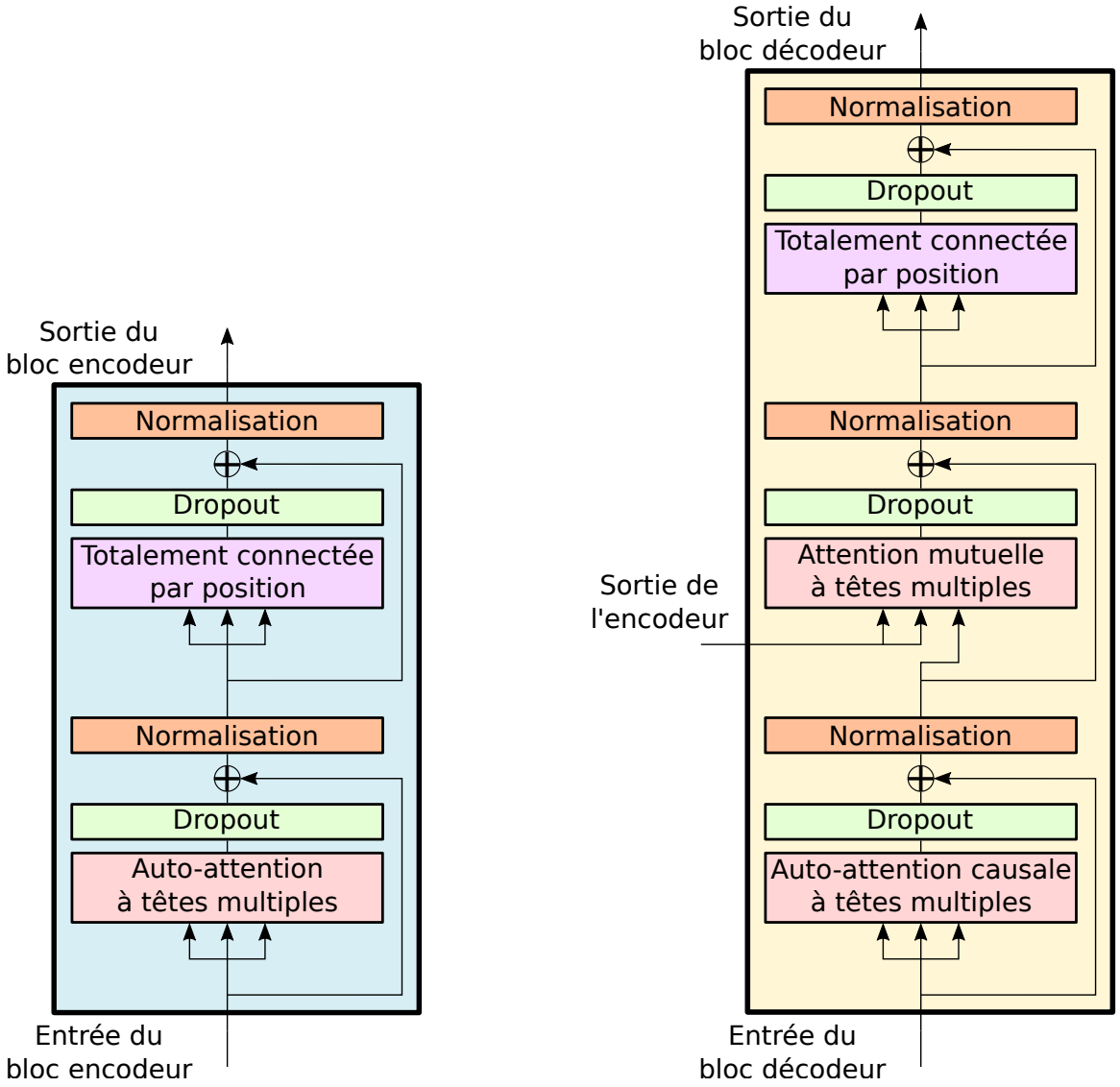
3.3 Blocs encodeur et décodeur

Un encodeur est basé sur un empilement de plusieurs blocs d'encodeur, et de même pour un décodeur. Ces blocs sont composés de mécanismes d'attention à têtes multiples, ainsi que de couches de neurones totalement connectés par position. Ces différents constituants ont été détaillés dans les sections précédentes. Dans cette section, nous détaillons la constitution des blocs d'encodeur et de décodeur, et nous expliquons leur fonctionnement.

3.3.1 Bloc encodeur d'une architecture Transformer

Un bloc d'encodeur classique d'une architecture Transformer (visible dans la figure 3.5a) est principalement composé de deux sous-blocs. Dans un premier temps, un sous-bloc d'auto-attention à têtes multiples permet de capturer des dépendances contextuelles entre les différentes parties de la séquence d'entrée du bloc. À la suite de ce premier sous-bloc, un second sous-bloc basé sur une couche de neurones totalement connectés par position permet de produire une représentation enrichie.

Du *dropout* est utilisé à la suite de chacun des sous-blocs, permettant d'effacer aléatoirement des valeurs des activations. Cet effacement permet notamment de régulariser l'entraînement. À la suite de cela, une connexion résiduelle [He+16] permet d'ajouter à la séquence les valeurs d'activation de la séquence du sous-bloc précédent. Cela offre un raccourci au gradient lors de l'étape de propagation inverse, et permet alors une conver-



(a) Exemple d'un bloc d'encodeur classique dans une architecture de Transformer.

(b) Exemple d'un bloc de décodeur classique dans une architecture de Transformer.

FIGURE 3.5 – Schéma d'un bloc d'encodeur (a) et d'un bloc de décodeur (b) typiquement utilisés dans les architectures de Transformer.

gence plus rapide. Une fonction de normalisation par couche [BKH16] est utilisée après la connexion résiduelle pour normaliser les activations générées par les différents sous-blocs.

3.3.2 Bloc décodeur d'une architecture Transformer

Un bloc de décodeur Transformer a pour séquence d'entrée soit la sortie du bloc décodeur précédent, soit la représentation vectorielle de la séquence d'entrée du décodeur (comme illustré dans la figure 3.1).

Les blocs de décodeur se décomposent en trois sous-blocs principaux : un sous-bloc d'auto-attention ; un sous-bloc d'attention mutuelle ; ainsi qu'un sous-bloc de neurones totalement connectés par position. Tout comme pour les blocs d'encodeur Transformer, chaque sous-bloc est accompagné de dropout, d'une connexion résiduelle et d'une normalisation. Une illustration d'un bloc de décodeur est présentée dans la figure 3.5b.

Dans un premier temps, un sous-bloc d'auto-attention causale à têtes multiples est utilisé. Cette attention a ainsi pour but de modéliser des dépendances contextuelles sur la séquence d'entrée du décodeur.

Deuxièmement, un sous-bloc d'attention mutuelle (ou encodeur-décodeur) à têtes multiples est utilisé. En particulier, ce sous-bloc utilise la sortie de l'encodeur pour générer les clés K et les valeurs V. Les requêtes sont quant à elles émises par le décodeur et cherchent une réponse dans la représentation riche produite par l'encodeur.

Enfin, un sous-bloc de neurones totalement connectés par position est utilisé pour produire une séquence de caractéristiques riches en informations.

3.4 Avantages et inconvénients des Transformers

Les architectures de Transformer offrent de nombreux avantages en comparaison à des architectures récurrentes tels que de meilleurs résultats sur diverses tâches [Dev+18] ou encore une meilleure parallélisation des calculs. En revanche, ces architectures possèdent certaines contreparties comme un besoin grandissant en données annotées.

3.4.1 Modélisation d'informations contextuelles

Le mécanisme d'attention à têtes multiples permet d'accéder à des informations contextuelles en un nombre constant d'opérations quelle que soit leur distance dans la séquence. En comparaison à des couches récurrentes, ces couches d'attention combinées avec de

l'encodage positionnel, permettent ainsi une bien meilleure modélisation du contexte, en particulier pour du contexte lointain.

En revanche, l'architecture Transformer doit en contrepartie être capable d'agréger l'ensemble des informations provenant de la séquence. La quantité d'informations à traiter en simultané, ainsi que l'impact sur la mémoire grandit alors quadratiquement avec la taille de la séquence. Il devient alors crucial de fournir suffisamment de données annotées pour contrer cette complexité accrue.

3.4.2 Vitesse d'apprentissage

Les architectures Transformer sont bien plus parallélisables en comparaison aux couches récurrentes. En effet, le mécanisme d'attention de ces architectures et autres couches utilisées dans les architectures Transformer ne nécessitent pas de traitement séquentiel des séquences traitées. Grâce à ce gain en vitesse d'apprentissage, les architectures à base de Transformer ont montré de bien meilleures capacités d'apprentissage ainsi que de biens meilleurs résultats [Dev+18].

Cependant, les couches d'attention causales présentes dans un décodeur suivent un processus autorégressif, et donc séquentiel. Vaswani et al. [Vas+17] proposent de paralléliser ce processus lors de la phase d'apprentissage. Pour cela, ils se basent sur l'utilisation d'un apprentissage forcé (*teacher forcing*) [FL90], combiné à un masque pour forcer la causalité de l'attention. Cependant, la phase de prédiction suit toujours un processus lent de décodage séquentiel. Diverses techniques ont pu être proposées pour réaliser un décodage parallélisable, mais au prix de moins bons résultats. Gu et al. [Gu+17] proposent par exemple pour la tâche de traduction automatique de réutiliser l'entrée de l'encodeur pour le décodeur, afin d'effectuer une traduction rapide. De plus, un apprentissage forcé peut créer un décalage entre la phase d'apprentissage et la phase de prédiction, où le modèle n'aurait pas appris à réagir à ses erreurs. Mihaylova et Martins [MM19] proposent par exemple un processus d'apprentissage du décodeur en deux étapes pour simuler des erreurs dans la vérité terrain et remédier à ce problème.

3.5 Discussions sur les Transformers

Les architectures Transformer représentent une excellente alternative aux couches récurrentes, grâce à de bonnes capacités de modélisation de la langue et d'apprentissage. Elles se sont rapidement imposées dans les diverses tâches du traitement automatique de

la langue [Dev+18]. Elles sont à l'état de l'art dans le domaine, avec des architectures de plus en plus larges pouvant atteindre des centaines de milliards de paramètres [Sca+22]. Néanmoins, ces architectures ne sont pas simples à entraîner, et nécessitent d'importantes quantités de données.

Elles ont aussi été appliqués dans d'autres domaines traitant des séquences tels que la reconnaissance de la parole [Chi+18 ; Zha+20].

De même, le domaine de la reconnaissance de texte manuscrit, de par sa nature à traiter des données séquentielles, semble adapté à l'utilisation d'architectures Transformer. Les couches de Transformer pourraient fournir de meilleures capacités de modélisation de contextes, ainsi qu'un entraînement plus efficace. De plus, les architectures Transformer ont démontré de très bons résultats dans des tâches de modélisation de langue [Dev+18]. Ces capacités pourraient fortement bénéficier au domaine de la reconnaissance d'écriture pour réduire les taux d'erreur.

Il est à noter qu'au commencement de cette thèse, il n'existait à notre connaissance pas de travaux publiés sur l'utilisation d'architectures Transformer pour la reconnaissance de texte. Ainsi, nos contributions présentées dans le chapitre 6 sont basées uniquement sur l'état de l'art présenté jusqu'ici.

Dans le domaine de la reconnaissance d'écriture manuscrite, les architectures Transformer ont cependant suscité de nombreux travaux très récemment. En particulier, ce champ de recherche a très fortement évolué durant le déroulement de cette thèse. Le chapitre 4 abordera ainsi les récentes avancées dues aux architectures Transformer dans le domaine de la reconnaissance d'écriture manuscrite.

Chapitre 4

État de l'art des Transformer pour la reconnaissance d'écriture manuscrite

Dans ce chapitre, nous présentons les récentes approches qui ont été proposées dans l'état de l'art pour aborder le problème de la reconnaissance d'écriture manuscrite avec des architectures Transformer. Notons que les approches abordées dans ce chapitre sont très récentes et ont en particulier été proposées après le début de cette thèse.

Ces approches proposent de remplacer les couches récurrentes traditionnellement utilisées par des couches de Transformer pour leurs divers avantages (meilleure modélisation du contexte, meilleure vitesse d'apprentissage, *etc.*). Elles démontrent aujourd'hui d'excellents résultats et représentent la majorité des approches récentes.

Les approches basées sur des couches de Transformer dans le domaine se basent sur deux types d'approches. Les réseaux de neurones convolutifs à Transformer proposent de combiner des couches de convolution avec des couches d'encodeur Transformer. Nous détaillons ces architectures dans la section 4.1. Les architectures de Transformer autorégressives que nous présentons dans la section 4.2 considèrent en plus l'utilisation d'un décodeur Transformer pour apporter des capacités de modélisation de la langue et réduire les taux d'erreur. Enfin, dans la section 4.3, nous discutons des stratégies de décodage utilisées par ces différentes approches, avant de conclure sur diverses discussions.

4.1 Réseaux de neurones convolutifs à Transformer

Les architectures de neurones convolutifs à Transformer sont des architectures qui sont peu traitées dans l'état de l'art [Dia+21 ; dAr+22]. Elles peuvent se résumer à une architecture traditionnelle de CRNN (présentée dans la section 2.2.3, p. 36), dans laquelle les couches récurrentes sont remplacées par des couches d'encodeur Transformer. Elles

combinent alors une base convolutive (*convolutional backbone*) pour l'extraction de caractéristiques, avec un encodeur Transformer pour intégrer des dépendances contextuelles. La figure 4.1 résume ces architectures.

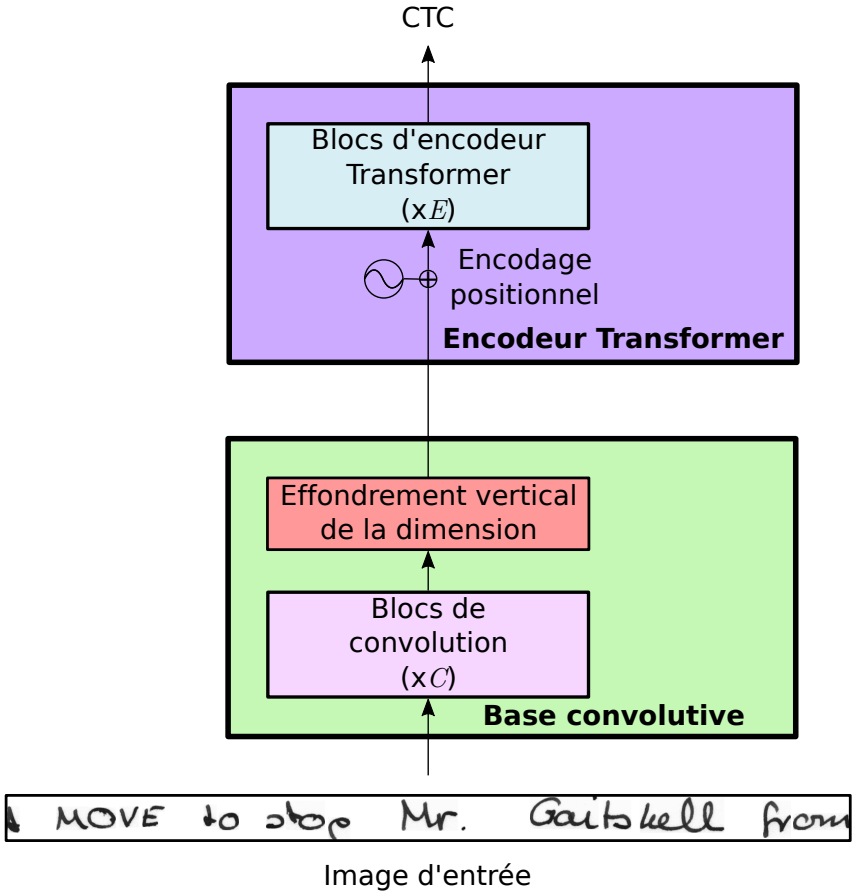


FIGURE 4.1 – Schéma classique d'un réseau de neurones convolutifs à Transformer. Le découpage des différentes parties de l'architecture reflète le plan de la section.

4.1.1 Base convolutive

De la même manière que pour les architectures présentées dans le chapitre 2, une base convolutive est utilisée pour extraire automatiquement des caractéristiques visuelles pertinentes pour la reconnaissance de caractère. Cette base de convolution est composée de plusieurs blocs (ou couches) de convolution, placés les uns à la suite des autres.

D'Arce et al. [dAr+22] utilisent une combinaison de 5 couches de convolution classiques dans leur base convolutive. Diaz et al. [Dia+21] utilisent quant à eux une base convolutive se servant de couches d'entonnoir inversé fusionnées (*Fused Inverted Bottle-*

neck Layers) [Xio+21]. Ces couches permettent en particulier une efficacité des calcul accrue sur des appareils portables.

Les cartes de caractéristiques obtenues sont converties en une séquence de caractéristiques, afin de permettre leur utilisation par des couches de Transformer.

4.1.2 Encodeur Transformer

Un encodeur Transformer est utilisé pour permettre d'intégrer et de résumer l'information contextuelle présente dans la séquence de caractéristique produite. Ces couches n'étant en revanche pas capables de distinguer l'emplacement relatif des caractéristiques traitées, une première étape d'encodage positionnel est alors réalisée (comme expliqué dans la section 3.1.3, p 46).

L'architecture proposée par D'Arce et al. [dAr+22] utilise un encodeur Transformer composé de 4 blocs d'encodeur Transformer avec 4 têtes d'attention et 192 neurones par couche au total. Il en résulte une architecture globalement légère, avec un total de 1,74M de paramètres.

La meilleure architecture proposée par Diaz et al. [Dia+21] est pour sa part composée de 16 blocs consécutifs d'encodeur Transformer avec 4 têtes d'attention et 256 neurones par couche. Elle est en revanche plus large, avec environ 12M de poids entraînables.

4.1.3 Avantages et limites

Les réseaux de neurones convolutifs à Transformer sont relativement légers en comparaison à d'autres architectures de Transformer. Ils s'entraînent rapidement [dAr+22] et peuvent être entraînés avec des quantités de données réduites.

Sans ajout d'un modèle de langue externe, l'architecture proposée par Diaz et al. [Dia+21] obtient d'excellents résultats en s'entraînant sur un jeu de données privé, avec un CER de 3,53% sur IAM, et de 2,48% sur RIMES. En revanche, les auteurs ne fournissent pas de résultats obtenus avec des architectures entraînées sans données privées. D'Arce et al. [dAr+22] obtiennent quant à eux un CER de 10,26% sur le jeu de données IAM. Ces résultats sont en revanche moins bons comparés à ceux obtenus par les autres approches de l'état de l'art.

Faute de résultats sans données privées pour l'approche présentée par Diaz et al. [Dia+21], il est difficile de conclure de manière précise sur l'intérêt de ces architectures. Diaz et al. [Dia+21] indiquent que leur architecture est capable d'obtenir d'excellents CER en

s'entraînant sur des données privées, tandis que D'Arce et al. [dAr+22] obtiennent de moins bons CER en n'utilisant pas de données extérieures. En ajoutant des données synthétiques, D'Arce et al. [dAr+22] obtiennent un CER de 7,50%. Cela reste encore inférieur aux autres approches de l'état de l'art, et peut ainsi indiquer que leur architecture n'est pas adaptée à la tâche.

Pour réduire les taux d'erreur de ces architectures, il peut être intéressant de considérer l'ajout de capacités de modélisation de la langue. L'ajout d'un décodeur Transformer pour intégrer de telles informations contextuelles semble une solution tout particulièrement indiquée, tandis que nous verrons dans la section 4.3 que l'architecture présentée par Diaz et al. [Dia+21] obtient de meilleurs taux d'erreur en ajoutant un modèle de langue additionnel.

4.2 Architectures de Transformer autorégressives

Les architectures de Transformer autorégressives utilisent pour leur part un décodeur Transformer comme point clé des approches [SK21 ; WZG21 ; Kan+22 ; Ria+22 ; WZG22 ; CCP23 ; Li+23]. Un encodeur, généralement similaire à ceux présentés dans la section 4.1 est utilisé pour extraire des caractéristiques visuelles dans le but de reconnaître des formes, et donc des caractères. Le décodeur a pour but de réduire les taux d'erreur en modélisant la langue. Le décodeur suit un processus de décodage autorégressif, ou, dans le cas de la reconnaissance d'écriture, caractère par caractère. La figure 4.2 présente un exemple typique d'architecture de Transformer autorégressive.

4.2.1 Extraction et encodage d'informations visuelles

Base convolutive

Tout comme les réseaux de neurones convolutifs à Transformer, la majorité des architectures de Transformer autorégressives utilisent des couches de convolution [SK21 ; WZG21 ; Kan+22 ; Ria+22 ; WZG22 ; CCP23]. Celles-ci permettent d'effectuer une extraction de caractéristiques pertinentes pour reconnaître des caractères.

En revanche, l'organisation, le type et le nombre de couches de convolutions utilisées dans les bases convolutives diffèrent d'une approche à une autre.

Une première solution basique, et inspirée des architectures de CRNN, consiste à utiliser un faible nombre de couches de convolutions. Ces couches de convolutions sont

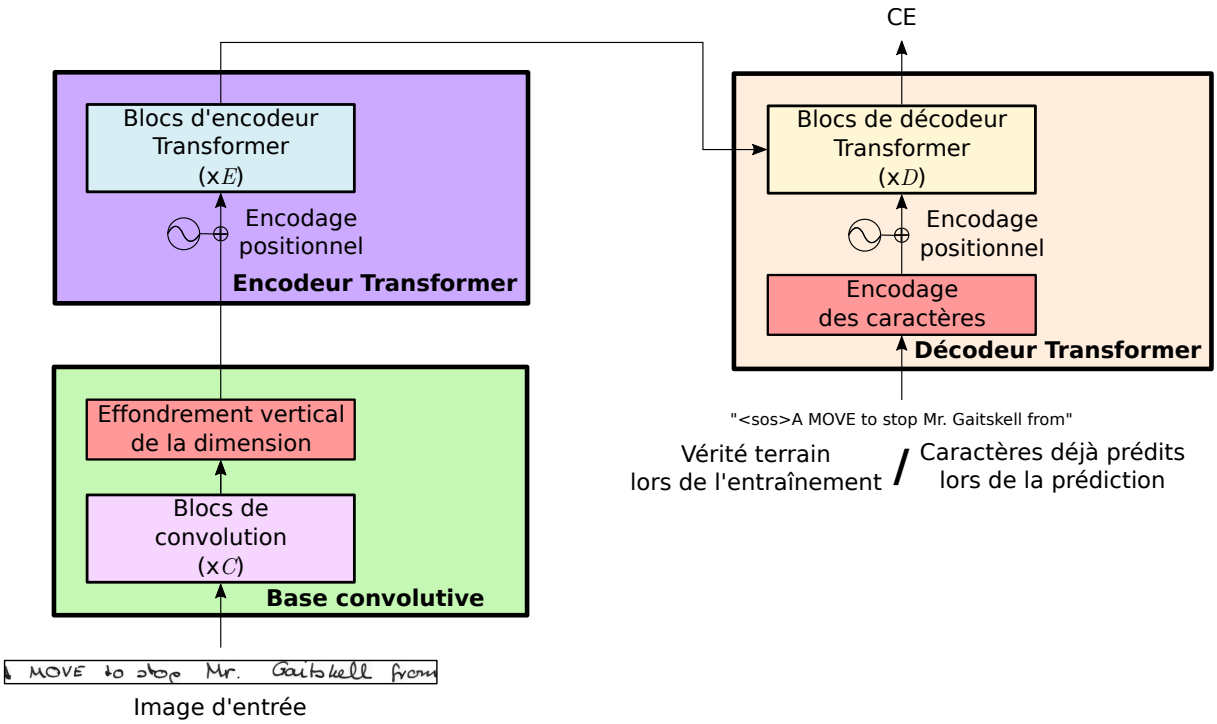


FIGURE 4.2 – Schéma classique d'une architecture de Transformer autorégressive pour la reconnaissance d'écriture. Le découpage des différentes parties de l'architecture reflète le plan de la section.

généralement en forme de goulot d'étranglement, réduisant la dimension verticale et horizontale des cartes de caractéristiques au fur et à mesure de l'application des couches. Le nombre de cartes de caractéristiques augmente à l'inverse. Wick et al. utilisent ainsi 3 couches de convolutions dans leurs architectures proposées [WZG21 ; WZG22], tandis que Riaz et al. [Ria+22] en utilisent un total de 7. Cette base convolutive démontre de bons résultats (4,15% sur IAM [WZG22]), tout en utilisant un nombre relativement restreint de poids.

D'autres approches considèrent des assemblages de couches de convolutions plus larges et complexes [SK21 ; Kan+22 ; CCP23]. Le nombre de poids entraînable étant plus important, des performances légèrement meilleures peuvent être attendues, généralement au prix d'une complexité d'entraînement accrue. Singh et al. [SK21], ainsi que Kang et al. [Kan+22] considèrent en particulier l'utilisation de réseaux de types ResNets [He+16], qui ont fait leurs preuves dans le domaine de la vision par ordinateur en tirant notamment parti de couches résiduelles. Coquenet et al. [CCP23] utilisent pour leur part un encodeur totalement convolutif (FCN), utilisé notamment dans l'architecture VAN [CCP22]. Ces bases convolutives sont en général plus larges et donc plus difficiles à entraîner. Cependant, elles sont nécessaires pour réaliser des tâches plus complexes, comme par exemple la

reconnaissance d'écriture manuscrite dans des pages complètes [SK21 ; CCP23]. De plus, la complexité des bases convolutives reste relativement faible en comparaison à celles des couches de Transformer.

Encodeur Transformer

Comme pour le précédent type d'architecture, une grande partie des architectures de Transformer autorégressives utilisent des couches d'encodeur Transformer pour permettre l'intégration de connaissances contextuelles sur les caractéristiques de la séquence. Dans les architectures proposées par Wick et al. [WZG21] ainsi que Riaz et al. [Ria+22], 3 couches d'encodeur Transformer sont alors utilisées. Kang et al. [Kan+22] utilisent pour leur part un encodeur bien plus large, composé de 4 couches d'encodeur Transformer qui utilisent chacune 1 024 neurones par couche. La taille de l'encodeur est en particulier bien plus grande que ceux des autres approches. Il est combiné à un ResNet50 [He+16] dans le but de permettre une meilleure reconnaissance des formes de caractères, probablement au prix d'un besoin en données annotées plus important.

Li et al. [Li+23] proposent pour leur part des encodeurs bien plus larges et complexes. À la place d'une base convolutive et d'un encodeur Transformer, les auteurs utilisent des architectures de Transformer pour la vision (ViT) [Dos+20]. En particulier, ils utilisent les architectures $\text{DeiT}_{\text{SMALL}}$ [Tou+21] (22M paramètres), $\text{BEiT}_{\text{BASE}}$ et $\text{BEiT}_{\text{LARGE}}$ [Bao+21] (86M et 307M paramètres respectivement), pour leurs différents modèles. Les Transformers pour la vision n'utilisent pas de couches de convolutions pour traiter une image de manière classique et considèrent à la place une séquence de blocs de pixels (ou *patchs*). Li et al. [Li+23] proposent un découpage de l'image en blocs de taille 16×16, avec une organisation de ces blocs dans l'ordre de lecture des écritures latines (de gauche à droite, puis de haut en bas). Leurs encodeurs étant particulièrement complexes, ils utilisent des modèles pré-entraînés sur des tâches de reconnaissance d'images, avant de les adapter à la tâche de reconnaissance d'écriture manuscrite. Ils montrent d'excellents résultats avec leur méthode (CER de 2,89% sur lejeu de données IAM), cependant, l'adaptation à la reconnaissance d'écriture reste une tâche particulièrement complexe compte tenu du nombre conséquent de paramètres utilisés. Un nombre important de données annotées est alors requis.

À l'inverse, Wick et al. [WZG22] utilisent une architecture de Transformer autorégressive basée sur un encodeur CRNN. Cette architecture ne bénéficie alors pas des différents avantages des couches de Transformer pour remplacer les couches récurrentes tels qu'une

meilleure modélisation du contexte ou encore un entraînement plus rapide. Nous pensons cependant que les auteurs ont fait ce choix dans le but de simplifier leur architecture et de la rendre plus simple à entraîner, tandis que leur contribution met l'accent sur un décodage différent.

Les approches Transformer utilisées par Singh et al. [SK21] ainsi que Coquenet et al. [CCP23], pour reconnaître de l'écriture au niveau d'un paragraphe ou d'une page complète, n'utilisent pas d'encodeur Transformer. Elles utilisent uniquement leur grand nombre de couches de convolution de la base convolutive pour reconnaître des caractères. Comme nous l'avons vu dans la section 2.4 (p. 39), des architectures FCN obtiennent de faibles taux d'erreur sans utiliser de couches récurrentes pour modéliser des dépendances contextuelles. Pour modéliser ce contexte, elles se reposent sur d'autres points comme par exemple de larges champs réceptifs. Tout comme des couches récurrentes, un encodeur Transformer n'est alors pas nécessaire, alors qu'il viendrait augmenter la complexité d'apprentissage. Les encodeurs des architectures de Singh et al. [SK21] et de Coquenet et al. [CCP23] sont alors suffisants pour modéliser des contextes à une échelle proche des caractères, tandis qu'un décodeur Transformer permet de modéliser des dépendances contextuelles larges.

4.2.2 Décodeur Transformer

Enfin, les différentes architectures utilisent une succession de couches de décodeur Transformer pour modéliser des dépendances contextuelles larges, diverses et complexes. Comme expliqué dans les sections 3.1 (p. 44) et 3.4.2 (p. 56), ces architectures autorégressives procèdent à un décodage successif. Les décodeurs Transformer utilisent en entrée : la séquence encodée (voir les sections 3.1.2 p. 45, et 3.1.3 p. 46) des caractères déjà produits ; ainsi que la sortie de l'encodeur du réseau. Ils produisent en sortie un caractère par étape du décodage. Comme expliqué dans la section 3.4.2 (p. 56), un processus d'entraînement forcé est utilisé pour grandement accélérer le processus à l'entraînement.

La plupart des architectures du domaine utilisent une structure de décodeur similaire, avec une succession de couches de décodeurs Transformer [SK21 ; WZG21 ; Kan+22 ; Ria+22 ; CCP23 ; Li+23]. Certaines architectures privilégient des décodeurs qui utilisent un nombre assez faible de paramètres, avec entre 3 et 6 couches de décodeurs Transformer composées de 256 neurones par couche [WZG21 ; Ria+22 ; CCP23] (260 pour Singh et al. [SK21]). De plus, le décodeur proposé par l'approche traitant des pages entières de Coquenet et al. [CCP23] se révèle particulièrement utile pour produire des informations

sur la structure de documents telles que la présence de colonnes, de paragraphes, ou encore de titres.

Wick et al. proposent dans de plus récents travaux [WZG22] un décodeur Transformer plus large, qui utilise 2 couches de décodeur Transformer composées de 8 têtes d'attention et de 512 neurones par couche. Kang et al. [Kan+22] utilisent une architecture plus large encore, avec 4 couches de décodeur Transformer, 8 têtes d'attention et 1 024 neurones par couche.

Li et al [Li+23] utilisent quant à eux des modèles de langue larges pré-entraînés en tant que décodeur de leur architecture, avant de les réentraîner. Dans leurs architectures proposées, ils utilisent les modèles MiniLM [Wan+20] (66M paramètres), ainsi que $\text{RoBERTa}_{\text{LARGE}}$ [Liu+19] (355M paramètres).

Enfin, il est à noter que les approches qui n'utilisent pas d'encodeur Transformer ajoutent de l'information positionnelle sur les séquences de caractéristiques produites par les encodeurs. Pour les approches traitant des paragraphes ou des pages complètes, un encodage positionnel 2D [Par+18] est alors ajouté pour inclure de l'information sur la position 2D dans l'image. En revanche, dans l'approche utilisée par Wick et al [Par+18], aucune information positionnelle n'est ajoutée à la suite de l'encodeur CRNN utilisé. À la place, les auteurs indiquent que l'architecture CRNN permet un apprentissage d'un tel encodage.

4.2.3 Discussions

Les architectures de Transformer autorégressives tirent parti d'un décodeur Transformer pour permettre une modélisation de la langue, et donc diminuer les taux d'erreur de telles architectures. Elles obtiennent d'excellents CER sur les jeux de données existants, comme par exemple des CER généralement inférieurs à 5% sur le jeu de données IAM [Kan+22 ; WZG22 ; CCP23 ; Li+23] et un CER de 3,49% sur RIMES [WZG22].

En revanche, ces architectures utilisent un nombre de poids entraînables bien plus important. Bien que cela permette une réduction des taux d'erreur, il est accompagné d'une augmentation importante en données annotées. L'architecture de Transformer présentée par Kang et al. [Kan+22] contient 100M de poids entraînables et obtient par exemple un excellent CER de 4,67% sur IAM en s'entraînant avec des données synthétiques, tandis que celui-ci chute à une valeur de 7,62% sans données synthétiques. L'architecture $\text{TrOCR}_{\text{LARGE}}$ est quant à elle composée de 558M de paramètres et atteint un CER de 2,89% grâce à des modèles pré-entraînés sur des données externes ainsi que l'utilisation de

nombreuses données supplémentaires. Aucun résultat n'est en revanche disponible pour un entraînement sans ces données. Compte tenu du nombre de paramètres très important et du nombre de données annotées des jeux de données de reconnaissance d'écritures, il peut être supposé que le modèle ne parviendrait pas à converger.

4.3 Modélisation de la langue et stratégies de décodage

4.3.1 Modélisation de la langue avec des couches Transformer

Dans les approches classiques de reconnaissance d'écriture manuscrite, une modélisation de la langue est fréquemment utilisée pour réduire les taux d'erreur [San+16 ; BM17 ; Pui17 ; Str+18 ; Sou+19]. (Cet aspect sera développé plus en détail dans la section 5.2.2, p. 79.) Dans le domaine du traitement automatique de la langue, des architectures à base de couches de décodeur Transformer sont particulièrement connues pour modéliser la langue de manière efficace.

Pour des réseaux de neurones convolutifs à Transformer qui n'utilisent pas de couches de décodeur Transformer, Diaz et al. [Dia+21] montrent une réduction importante du CER sur IAM de 3,53% à 2,75% en utilisant un modèle de langue externe. Cela indique en particulier que des couches d'encodeur Transformer ne sont pas suffisantes pour modéliser la langue.

Pour une architecture basée sur un décodeur Transformer et un décodage autorégressif, Kang et al. [Kan+22] n'observent pas de différence significative en utilisant un modèle de langue externe (4,66% au lieu de 4,67% sur IAM). Les récents travaux de Wick et al. [WZG22], indiquent pour leur part une détérioration du CER de 4,15% sur IAM à 6,46% en utilisant un modèle de langue pour leur architecture. Ces résultats semblent indiquer que les architectures de Transformer autorégressives sont adaptées pour modéliser la langue de manière efficace et réduire les taux d'erreur.

4.3.2 Stratégies de décodage existantes

Pour des réseaux de neurones convolutifs à Transformer, l'étape de décodage des caractères peut être réalisée en parallèle. Ce processus a l'avantage d'être particulièrement rapide. Pour des architectures de Transformer autorégressives, un processus de décodage

étape par étape similaire à des couches récurrentes est nécessaire. Cela ralentit fortement l'étape de prédiction, mais permet en revanche de faibles taux d'erreur.

Wick et al. [WZG21] proposent un processus de décodage différent, reprenant le principe des couches récurrentes bidirectionnelles (présentées dans la section 2.2.1, p. 33). Ils utilisent pour cela deux décodeurs Transformer : l'un réalise une étape de prédiction dans le sens de la lecture ; tandis que l'autre réalise une prédiction dans le sens inverse de la lecture. Les prédictions sont ensuite combinées au moyen d'un processus de vote. Grâce à un deuxième décodeur Transformer, les auteurs réduisent le CER de 6,03% à 5,67% sur IAM. Le nombre de paramètres utilisé passe en revanche de 13M à 27M, mais la complexité de l'entraînement reste cependant similaire.

Dans une contribution plus récente [WZG22], Wick et al. proposent de combiner un décodage des caractères étape par étape produit par le décodeur Transformer avec un décodage de la sortie du décodeur basé sur l'approche CTC. De plus, un modèle de langue peut aussi être combiné avec ces deux premières approches. Cette stratégie se montre bénéfique et permet une amélioration du CER de 4,15% à 3,96% sur IAM, ainsi que d'atteindre un CER de 3,13% en ajoutant un modèle de langue externe.

Enfin, pour obtenir la séquence de sortie produite par l'architecture à partir des séquences de probabilités, deux approches s'opposent. Un décodage par faisceau (*beam search decoding*) est fréquemment utilisé pour apporter une séquence avec moins d'erreurs, au prix d'un temps de décodage plus important. Cette approche est considérée par différentes approches classiques de reconnaissance d'écriture, ainsi que des approches de Transformer [dAr+22 ; Ria+22 ; WZG22 ; Li+23]. Cependant, diverses approches démontrent qu'un décodage par faisceau n'est pas utile [Dia+21 ; SK21 ; Kan+22] et qu'une approche de décodage gloutonne en prenant le meilleur chemin à chaque étape (*best-path decoding*) fournit des résultats similaires.

4.4 Discussions sur les architectures Transformer en reconnaissance d'écriture

Les différentes approches de Transformer qui ont été très récemment proposées dans le domaine montrent de très bons résultats. Ces améliorations peuvent notamment s'expliquer par les capacités des couches Transformer à modéliser efficacement diverses informations contextuelles.

Une tendance pour réduire les taux d'erreur consiste à augmenter la largeur de ces architectures. En revanche, le nombre de données annotées dans le domaine reste toujours restreint, et très vite limitant. Il le reste en particulier même pour des réentraînement de modèles pré-entraînés.

Plusieurs possibilités peuvent être envisagées pour obtenir de bons résultats avec de telles architectures. D'une part, l'ajout de données annotées peut être considéré. Si certaines approches de l'état de l'art considèrent l'utilisation de données privées pour s'entraîner, cette solution n'est cependant pas envisageable pour tout le monde. La génération de données synthétiques se révèle quant à elle être une excellente solution à ce problème, puisqu'elle permet la génération de données annotées à moindre coût. Nous aborderons notamment cet aspect dans le chapitre 5.

D'autre part, il peut être intéressant de proposer des architectures de Transformer à la fois performantes pour obtenir de faibles taux d'erreur, mais aussi légères et efficaces pour s'entraîner avec peu de données annotées. Cette solution représente en particulier un axe de recherche important des architectures de Transformer que nous proposons en tant que contribution dans les chapitres 6 et 7.

Enfin, nous rappelons que les approches décrites dans ce chapitre ont été proposées au cours du déroulement de cette thèse. Les premiers travaux réalisés, et proposés dans le chapitre 6 ne se basent donc pas sur le contenu de ce chapitre. En revanche, elles ont été considérées dans les réflexions sur les chapitres suivants au fur et à mesure de leur publications.

Chapitre 5

Documents anciens et manque de données

Les architectures de réseaux de neurones les plus récentes permettent d'obtenir de meilleurs résultats, mais demandent en contrepartie une grande quantité de données annotées. Cependant, dans le domaine de la reconnaissance d'écriture, les données annotées sont bien souvent disponibles en quantités trop faibles pour ces architectures. C'est en particulier le cas des documents anciens que nous visons à reconnaître, qui, en plus d'être difficiles à reconnaître, possèdent très peu de données annotées.

Ce chapitre se concentre sur les différentes difficultés de reconnaissance, liées aux documents anciens et au manque de données annotées. La section 5.1 présente les problématiques liées aux documents anciens, et introduit des jeux de données existants. Dans la section 5.2, nous présentons diverses approches de l'état de l'art abordant la reconnaissance de documents, ainsi que leur différentes méthodes. Enfin, dans la section 5.3, nous nous intéressons en particulier au problème général du manque de données, et aux solutions existantes concernant des documents modernes ou anciens.

5.1 Documents anciens

La notion de document ancien fait référence à un ensemble de documents sur une large période temporelle. Dans le cadre de cette étude, les documents anciens que nous visons à traiter s'étendent entre le XV^{e} siècle et le XIX^{e} siècle.

En comparaison à des documents modernes, les documents anciens sont bien plus complexes à cause de nombreuses difficultés qui leurs sont propres, et dont nous discuterons ci-dessous. Cela affecte particulièrement les annotateurs humains, mais aussi les systèmes de reconnaissance d'écriture manuscrite qui peinent à obtenir des taux d'erreur acceptables sur les documents les plus complexes. En général, un CER de l'ordre de 10%

est considéré comme acceptable [Str+18], car cela permet à un annotateur humain de corriger les erreurs restantes, avec un effort réduit.

Dans la section 5.1.1, nous abordons les différentes difficultés de reconnaissance liées aux documents anciens. Par la suite, la section 5.1.2 présente différents jeux de données anciens existants, ainsi que leurs particularités.

5.1.1 Difficultés des documents anciens

Variabilité des documents

Les écritures dans des documents anciens sont composées d'une encre très variée, avec en plus des couleurs d'encres classiques, de l'encre avec une teinte marron, due à la dégradation de ses composés au fil du temps. Les fonds des documents anciens sont eux aussi très hétérogènes, avec une grande variété dans la nature, la teinte, et la granularité des papiers utilisés. Ces différentes variations s'illustrent par exemple dans la figure 5.5 (p. 77).

Les documents anciens que nous traitons s'étendent sur une période assez large. Cela peut ainsi résulter en différentes évolutions de la langue ou du style d'écriture. L'apprentissage d'un modèle de langue pour traiter un document spécifique est alors complexe.

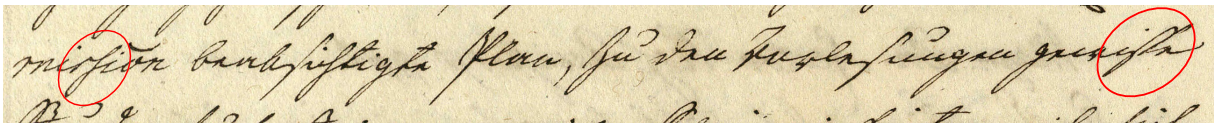
Écritures complexes

Les documents anciens contiennent généralement des écritures complexes, avec par exemple certaines formes de caractères complexes. C'est le cas en particulier, du caractère "`s` long", à la forme proche du caractère "`f`", dont dérive aujourd'hui le caractère "`s`". Cette forme de caractère est principalement utilisée dans les documents antérieurs au XIX^{e} siècle. Ce caractère est illustré dans la figure 5.1a.

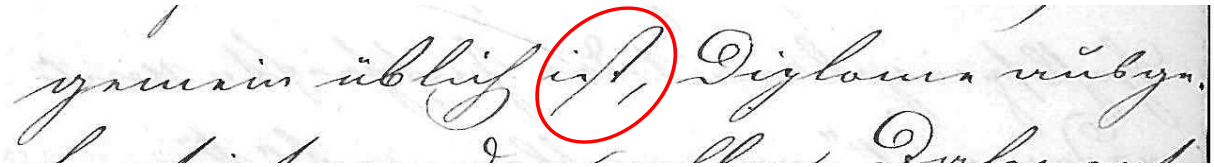
De plus, d'autres formes de caractères complexes peuvent exister selon le contexte entourant un caractère. C'est le principe des ligatures donnant des formes contextuelles aux différents caractères, et en particulier dans les documents anciens pour les caractères "`s`" ou "`f`". Des exemples de ligatures historiques sont présentés dans les figures 5.1b et 5.1c.

Enfin, la reconnaissance d'une ligne de texte peut être compliquée par la présence d'enjambements. Dans ce cas, les caractères descendants de la ligne de texte précédente, ainsi que les caractères ascendants de la ligne de texte suivante peuvent alors apparaître

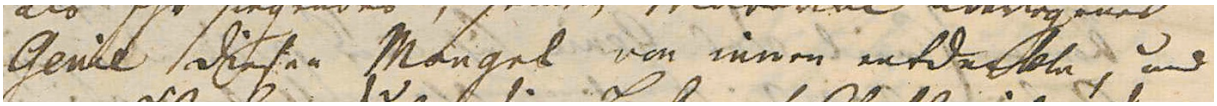
(a) Exemple d'écriture ancienne utilisant la forme de caractère "s long" à de multiples reprises, ainsi qu'une écriture fortement inclinée. Vérité terrain associée : "`Bis dahin verspare ich so manches das ich`".



(b) Exemple de ligatures sur le bigramme "`ss`". Les ligatures sont entourées en rouge sur l'image. Vérité terrain associée : "`mission beabsichtigte Plan, zu den Vorlesungen gewisse`".



(c) Exemple de ligature sur le bigramme "`st`". Celle-ci est entourée en rouge sur l'image. Vérité terrain associée : "`gemein ublich ist, Diplome ausge-`".



(d) Exemple de ligne de texte avec un enjambement prononcé de la ligne précédente, ainsi qu'avec des écritures présentes au verso de la page qui transparaissent.

FIGURE 5.1 – Exemples d'écritures complexes avec : (a) des formes de caractères anciennes ; (b) et (c) des ligatures anciennes ; et (d) un enjambement marqué. Les images sont extraites du jeu de données de la compétition READ 2018 [Str+18].

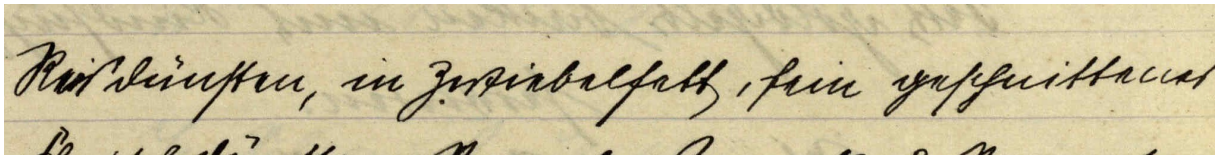
sur une ligne de texte et ainsi empiéter sur la graphie de certains caractères. Cela peut alors gêner les modèles de reconnaissance de caractères. La figure 5.1d illustre ce principe.

Dégradation des documents

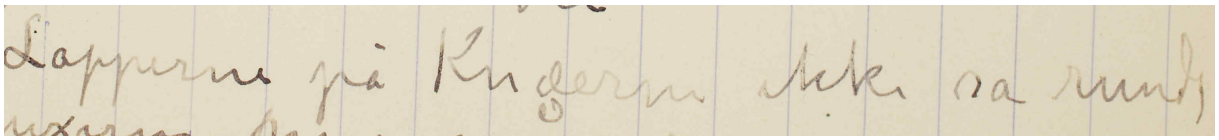
D'autres difficultés liées à des dégradations dues au passage du temps sont fréquentes. Dans des conditions de conservation défavorables, le papier peut se retrouver fragilisé. Il s'effritera ou se déchirera plus facilement, provoquant une perte d'une partie du texte.

En plus d'éventuelles taches, l'encre peut elle aussi subir différentes dégradations dues au passage du temps telles que : une teinte de l'encre virant au marron ; un effacement plus ou moins fort de l'encre ; un traversement de l'encre qui peut alors transparaître sur le

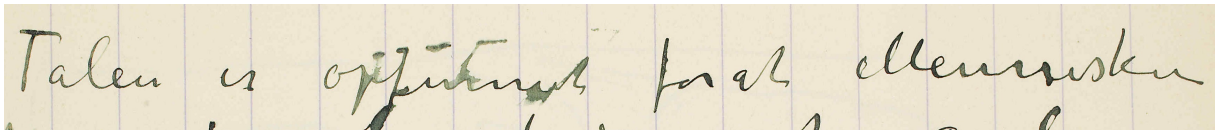
verso d'une page ; ou un étalement de l'encre. Ces différentes détériorations peuvent alors affecter la reconnaissance du document. La figure 5.2 illustre ces diverses dégradations.



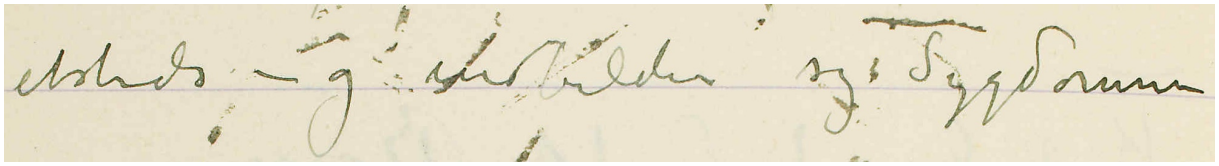
(a) Exemple d'encre traversant la page.



(b) Ligne de texte contenant de l'encre partiellement effacée.



(c) Image avec un étalement de l'encre.



(d) Exemple de texte complexe avec de multiples détériorations de l'encre.

FIGURE 5.2 – Différents exemples de dégradations de l'encre sur divers documents. Les images sont extraites du jeu de données de la compétition READ 2018 [Str+18].

5.1.2 Jeux de données anciens

Dans cette section, nous présentons des jeux de données publics de l'état de l'art abordant la reconnaissance d'écriture manuscrite dans des documents anciens.

Jeu de données READ 2016

Le jeu de données READ 2016 [San+16] est un regroupement de comptes-rendus écrits de réunions de conseils extraits de la collection Ratsprotokolle. Ces documents sont écrits en allemand, et s'étalent sur la période comprise entre 1470 et 1805. Le jeu de données

est assez homogène dans sa structure avec un bloc de texte principal sur chaque page, d'éventuelles annotations marginales, ainsi qu'un numéro de page. Le jeu de données fournit diverses images de pages, avec l'information de la position des boites englobantes de chaque ligne ainsi que du texte associé. Un exemple de pages est montré dans la figure 5.3.

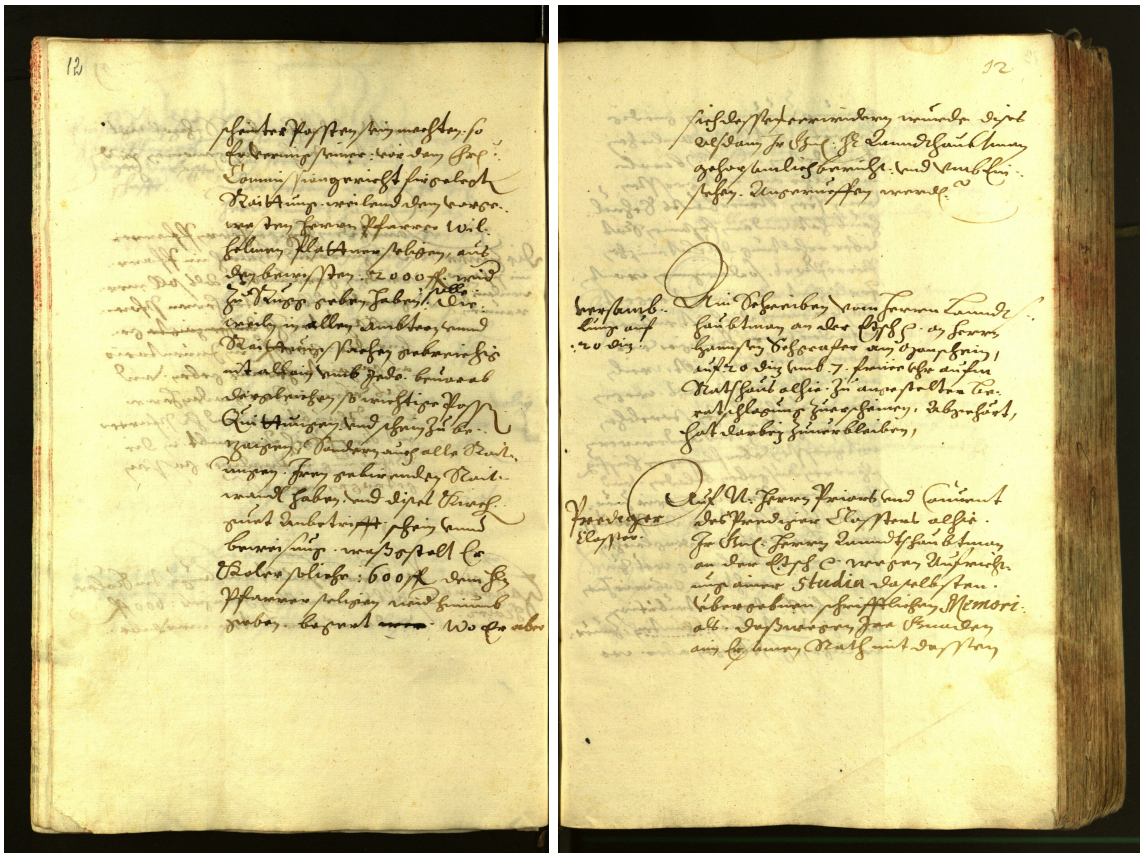


FIGURE 5.3 – Exemples de pages extraites du jeu de données READ 2016 [San+16].

Le jeu de données est séparé en trois partitions : la partition d'entraînement contient 350 pages (9 967 lignes) ; la partition de validation 50 pages (1 219 lignes) ; et 50 pages (1 335 lignes) pour la partition de test. Lors de la compétition qui a eu lieu sur le jeu de données, les auteurs ont proposé deux sous-compétitions. La première est restrictive et n'autorise uniquement que les données fournies pour s'entraîner, tandis que la deuxième autorise l'entraînement avec des données annotées extérieures.

Le jeu de données READ 2016 offre des données homogènes dans leur style en quantité assez importante pour la plupart des modèles. Les meilleures approches de l'état de l'art atteignent un CER proche de 4% au niveau ligne [CCP22 ; CCP23]. Ce jeu de données

permet en particulier de s'évaluer dans les cas où une quantité importante de données est disponible pour un document. Cependant, ce scénario n'est pas représentatif de la majorité des cas.

Jeu de données READ 2018

Le jeu de données READ 2018 [Str+18] a été proposé à l'occasion d'une compétition pour évaluer la capacité des systèmes de reconnaissance d'écriture manuscrite ancienne à se spécialiser sur des données spécifiques.

Le jeu de données READ 2018 se divise en deux parties principales : la partition d'entraînement générale ; ainsi que les divers documents de spécialisation. Il simule un scénario dans lequel des données annotées de divers documents sont disponibles (la partition d'entraînement générale), tandis que l'on souhaite transcrire intégralement un jeu de données spécifique (la partition spécifique), écrit en général par un unique auteur. Pour ce faire, on souhaite entraîner un modèle générique sur la partition d'entraînement générale, avant de spécialiser le modèle avec quelques exemples annotés du document spécifique. Un schéma résumant ce processus d'entraînement est disponible dans la figure 5.4.

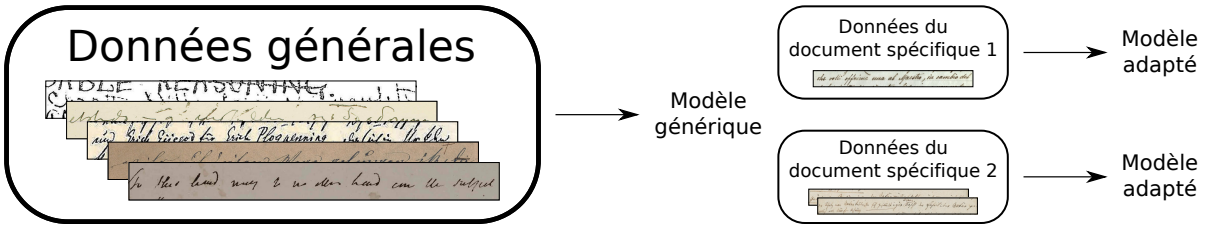


FIGURE 5.4 – Schéma explicatif du processus d'entraînement sur le jeu de données READ 2018. Un modèle générique est appris sur des données diverses, avant de spécialiser ce modèle sur un document spécifique parmi cinq proposés.

Partition d'entraînement générale Cette partition contient 11 903 lignes de texte annotées. La figure 5.5 montre diverses lignes de cette partition. La partition est composée de 17 documents variés et dans diverses langues (allemand, anglais, suédois, et danois). Bien que cette partition contienne un nombre de données annotées conséquent pour un jeu de données ancien, nous verrons que cela reste tout de même faible compte tenu de la difficulté de la tâche.

Documents de spécialisation Le jeu de données READ 2018 propose 5 documents spécifiques : Konzilprotokolle C ; Schiller ; Ricordi ; Patzig ; et Schwerin. Tandis que tous

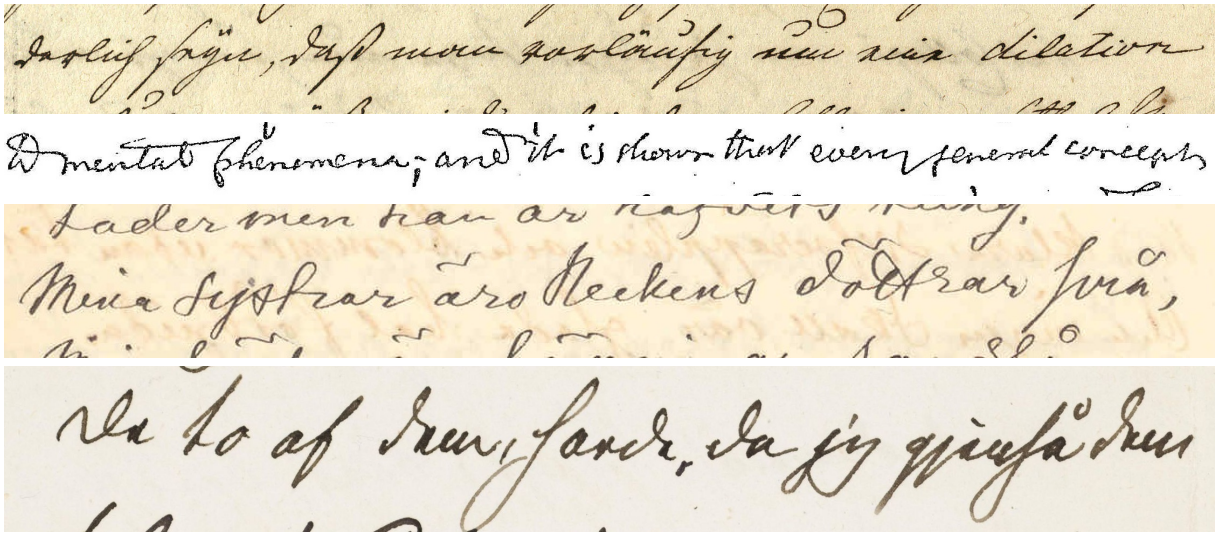


FIGURE 5.5 – Exemples de lignes de textes provenant du jeu de données de la compétition READ 2018 [Str+18]. Plus précisément, ces images proviennent de la partition d'entraînement générale du jeu de données. De haut en bas, ces lignes sont écrites en : allemand ; anglais ; suédois ; ainsi qu'en danois.

les autres documents spécifiques sont écrits en allemand, le document spécifique Ricordi est quant à lui rédigé en langue italienne. La spécialisation d'un modèle sur Ricordi est alors particulièrement complexe, car aucun document écrit en italien n'est présent dans la partition d'entraînement générale. Konzilprotokolle C reste quant à lui proche de deux documents présents dans la partition d'entraînement générale (Konzilprotokolle A et Konzilprotokolle B). Une illustration des documents spécifiques est disponible dans la figure 5.6.

La compétition propose d'évaluer la qualité de la reconnaissance en fournissant pour chaque document spécifique quatre scénarios avec un nombre de pages annotées différent : 0 page annotée ; c'est-à-dire qu'aucune spécialisation sur le document spécifique n'est réalisée ; 1, 4, et enfin 16 pages annotées. Il est cependant à noter que 16 pages correspondent tout de même à un nombre de lignes annotées relativement faible (entre 250 et 750 lignes de texte annotées).

Bilan sur READ 2018 Le jeu de données READ 2018 offre un scénario complexe dans lequel il est à la fois important pour une architecture d'apprendre sur plusieurs documents variés, mais aussi d'être capable de bien généraliser et de se spécialiser avec très peu de données annotées. La meilleure approche de l'état de l'art obtient un CER moyen sur l'ensemble des documents de spécialisation et des scénarios de 13,02% [YHM20]. Il reste en particulier une marge de progression importante sur les différents scénarios proposés.

FIGURE 5.6 – Exemples de lignes des différents documents spécifiques de la compétition READ 2018 [Str+18]. De haut en bas, ces lignes proviennent des documents suivants : Konzilprotokolle C ; Schiller ; Ricordi ; Patzig ; et Schwerin.

5.2 Architectures de réseaux de neurones pour les documents anciens

Dans cette section, nous présentons comment les différentes architectures de réseaux de neurones abordent les différentes problématiques posées par les documents anciens.

5.2.1 Entraînement avec peu de données annotées

Les documents anciens contiennent généralement peu de données annotées (de l'ordre de quelques centaines de lignes annotées [Str+18]) à cause de nombreuses difficultés de reconnaissance évoquées précédemment, ainsi que des coûts d'annotations importants. À l'opposé, les réseaux de neurones nécessitent des quantités de données annotées proportionnellement à la difficulté des documents. Cela peut alors limiter l'apprentissage de réseaux de neurones, et en particulier les plus larges.

Les architectures type CRNN sont la plupart du temps utilisées pour reconnaître les documents anciens que ce soit sur le jeu de données READ 2016 [San+16] ou sur READ 2018 [Str+18 ; Sou+19]. Ces architectures utilisent assez peu de poids (généralement entre 1 et 2M) et sont donc capables de s'entraîner efficacement sur peu de données annotées. Sans modèles de langues ajouté, elles obtiennent un CER 5,1% sur le jeu de données READ 2016 [San+16], ainsi qu'un CER moyen proche de 17,9% sur READ 2018 [Str+18].

Des architectures FCN [YHM20 ; CCP21 ; CCP22] ont aussi été proposées pour reconnaître des documents anciens. Coquenet et al. [CCP21 ; CCP22] proposent ainsi des architectures FCN pour la reconnaissance de paragraphes ou de pages, ainsi qu'applicables à la reconnaissance de lignes. Sur le jeu de données READ 2016, l'architecture VAN [CCP22] utilise 2,7M paramètres et est capable d'obtenir un CER de 4,10% au niveau ligne et de 3,59% au niveau paragraphe. L'architecture SPAN [CCP21] possède quant à elle, 19,2M paramètres (7 fois plus) et obtient un taux d'erreur paragraphe de 6,20%. Cette différence notable peut en partie s'expliquer par un grand nombre de poids. De plus, très récemment, les auteurs ont égalé leurs précédents résultats au niveau ligne en intégrant leur FCN au sein d'une architecture type Transformer [CCP23] possédant 7,6M paramètres. Yousef et al. [YHM20] proposent quant à eux une architecture FCN dans le but d'être entraînée efficacement avec les données annotées disponibles. Celle-ci se base sur de nombreuses optimisations ainsi qu'une grande quantité d'augmentation de données. Néanmoins, avec 3,4M de paramètres, elle obtient un CER moyen de 13,02% sur READ 2018, ce qui représente le meilleur score global parmi les approches de l'état de l'art.

5.2.2 Modélisation de la langue

L'apprentissage de la langue joue un rôle important dans la reconnaissance d'écriture manuscrite ancienne. De nombreuses approches de l'état de l'art [San+16 ; Str+18 ; Sou+19] considèrent l'utilisation d'un lexique, ou d'un modèle de langue pour corriger d'éventuelles erreurs produites par le modèle optique, et améliorer les résultats. Par exemple, des architectures CRNN obtiennent, grâce à l'utilisation d'un modèle de langue, des CER aux alentours de 5% sur le jeu de données READ 2016 [San+16].

Pour modéliser la langue, la majorité des approches utilisent des modèles statistiques N-grammes de mots ou de caractères [San+16 ; Str+18]. Dans les différents travaux du LITIS [Swa+17 ; Str+18 ; Sou+19], les auteurs proposent à la place l'utilisation de multi-grammes [Swa+17], c'est-à-dire des séquences de caractères de longueur variable. De plus,

ils proposent de combiner un modèle de langue générique et multilingue, avec un modèle de langue spécifique, appris sur un document spécifique à un auteur [Str+18 ; Sou+19], dans le but de fournir une modélisation de la langue prenant en compte le nombre de données annotées disponibles. Ils obtiennent un CER moyen 14,56% sur le jeu de données READ 2018 [Sou+19].

La modélisation de la langue et du contexte peut aussi être réalisée de manière implicite dans les réseaux de neurones. Michael et al. [Mic+19] proposent ainsi d'intégrer un CRNN dans une architecture utilisant un mécanisme d'attention et un décodeur récurrent. Leur architecture *seq2seq* obtient ainsi un CER de 4,66% sur READ 2016.

Bien que les architectures FCN puissent être capables de modéliser des dépendances contextuelles, l'apprentissage de ces dépendances et de la langue par ces architectures est loin d'être trivial. Coquenet et al. combinent une architecture FCN avec des mécanismes d'attention, et observent un gain important [CCP22 ; CCP23] comparé à un FCN seul [CCP21]. Additionnellement, ils obtiennent de bien meilleurs résultats au niveau page, notamment grâce à l'apport d'informations contextuelles supplémentaires présentes dans l'enchaînement des différentes lignes de texte consécutives d'un paragraphe.

5.2.3 Discussions

La reconnaissance d'écriture manuscrite dans des documents anciens et complexes reste un problème difficile sur lequel peu de travaux ont été proposés. Pour résoudre cette tâche, il semble très important d'utiliser des architectures de réseaux de neurones adaptées à ces documents, avec notamment un nombre faible de poids, ainsi que de bonnes capacités de modélisation de la langue.

Les architectures Transformer, pourraient fortement bénéficier à la problématique de reconnaissance de documents manuscrits anciens. Elles offrent de bonnes capacités de modélisation d'informations contextuelles, et excellent dans l'apprentissage de la langue.

Cependant, les documents anciens contiennent généralement peu de données (de l'ordre de 100 à 1 000 lignes annotées, voire plus rarement, 10 000 pour les jeux de données les plus fournis ou composés de plusieurs documents). Cela impacte ainsi l'entraînement de l'ensemble des architectures, et plus particulièrement les architectures Transformer qui nécessitent des quantités de données annotées importantes. Il est alors crucial d'utiliser diverses stratégies pour faciliter l'apprentissage de tels réseaux, et pour augmenter la variabilité des données d'entraînement.

5.3 Traitements des données

Pour combler le manque de données annotées, les approches de reconnaissance d'écritures manuscrites ont principalement considéré trois stratégies différentes.

— Le pré-traitement des données, dans le but de réduire la variation entre des données annotées réelles. Nous en discuterons dans la section 5.3.1.

— L'augmentation de données annotées réelles pour augmenter la variabilité dans les images en entrée et ainsi mieux généraliser. Cet aspect sera abordé dans la section 5.3.2.

— La génération de données synthétiques, qui permet de générer de nouvelles données annotées. Les section 5.3.3 et 5.3.4 présenterons la génération de données synthétiques au style de documents modernes et anciens respectivement.

Ces trois stratégies indépendantes peuvent être appliquées seules ou en combinaison, et permettent ainsi d'améliorer les performances des réseaux de neurones.

5.3.1 Pré-traitements des données

Une première stratégie consiste à utiliser des algorithmes de pré-traitements pour normaliser les différentes variations présentes dans un ensemble d'images. Les différents modèles s'entraînent ainsi sur des données plus proches les unes des autres et peuvent donc se concentrer sur l'apprentissage de caractéristiques utiles à la reconnaissance d'écriture.

Les pré-traitements peuvent affecter l'image en elle-même en normalisant par exemple : sa taille ; les valeurs des pixels ; la luminosité ; ou encore le contraste de l'image. D'autres pré-traitements utilisent des connaissances sur la structure des lignes de texte et normalisent la courbure de la ligne [Pui17 ; Dut+18] ou l'inclinaison des caractères.

Cependant, ces méthodes possèdent certains inconvénients. En cas d'échec d'un algorithme de pré-traitement, les résultats finaux peuvent alors être dégradés. De plus, ces méthodes de normalisation limitent la capacité des réseaux de neurones à généraliser sur des données diverses. Au contraire, un modèle utilisant des caractéristiques génériques aura tendance à être plus performant sur de nouvelles données.

Pour ces raisons, la plupart des algorithmes de pré-traitements ne sont ainsi pas considérés dans les approches récentes, au profit d'augmentations de données. En pratique, seule une normalisation de la hauteur de l'image, ainsi qu'une normalisation centrée réduite des pixels de l'image sont encore considérés.

5.3.2 Augmentations de données traditionnelles

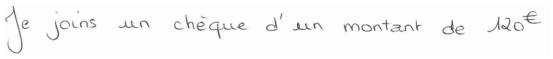
Pour palier le manque de données annotées, en particulier pour des documents anciens, les approches de reconnaissance d'écriture manuscrite considèrent très majoritairement l'utilisation d'augmentation de données [Pui17 ; Str+18 ; Sou+19 ; WZG21 ; Kan+22 ; SK21 ; CCP22 ; WZG22 ; dAr+22 ; CCP23 ; Li+23]. Certaines approches reposent en particulier sur une utilisation bien plus importante de transformations pour obtenir de bons résultats [YHM20 ; YB20].

Les techniques d'augmentation de données consistent à créer de nouvelles images à partir d'images réelles, en y appliquant une ou plusieurs déformations aléatoires. L'ensemble des labels des images augmentées reste quant à lui inchangé. Cela permet ainsi de virtuellement agrandir la taille du jeu de données en proposant de nouvelles images, proches d'exemples réels. En conséquence, un modèle pourra être entraîné à voir plus de variété dans ses données d'entraînement, et donc à limiter le phénomène de sur-apprentissage.

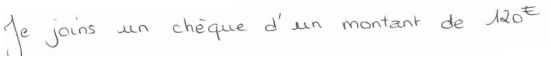
Augmentations utilisées en reconnaissance d'écriture manuscrite

Pour transformer une image, différentes augmentations sont en général choisies aléatoirement parmi une liste. L'ensemble des transformations est alors appliqué, avec pour chaque transformation, un ensemble de paramètres aléatoires. Les différentes transformations couramment utilisées dans le domaine de la reconnaissance d'écriture manuscrite peuvent se décliner en trois principales catégories : les transformations géométriques affines ; les augmentations de l'écriture ; ainsi que les transformations des pixels de l'image. Ces transformations sont expliquées ci-dessous, ainsi qu'illustrées par la figure 5.7.

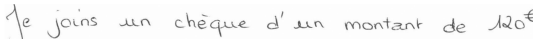
Transformations géométriques affines La grande majorité des approches de la reconnaissance d'écriture manuscrite utilise des transformations géométriques affines pour augmenter leurs données. Ainsi, les approches récentes [Dut+18 ; Sou+19 ; Mic+19 ; YHM20 ; SK21 ; CCP22 ; Kan+22] utilisent différentes transformations parmi les suivantes : rotation de l'image ; translation de l'image ; changement d'échelle. Quelques approches [Dut+18 ; Str+18 ; Sou+19 ; Kan+22] utilisent en particulier du cisaillement aléatoire sur les images. Ce cisaillement permet d'appliquer une inclinaison uniforme des caractères de l'image, par exemple pour appliquer un effet d'écriture italique, qui semble particulièrement pertinent pour de la reconnaissance d'écriture manuscrite. Yousef et al. [YHM20] utilisent dans leur approche une transformation pour modifier la perspective de l'image. Cette transformation permet d'obtenir des lignes de textes avec d'importantes



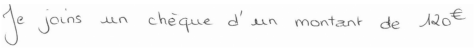
(a) Image originale



(b) Rotation aléatoire



(c) Translation aléatoire



(d) Changement d'échelle

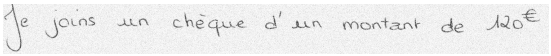
(e) Cisaillement de l'image

(f) Changement de la perspective

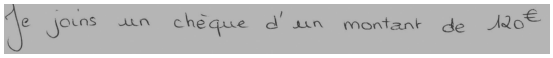
(g) Érosion de l'écriture

(h) Dilatation de l'écriture

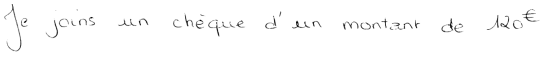
(i) Distorsions élastiques



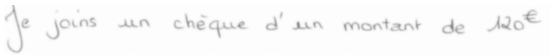
(j) Bruit ajouté



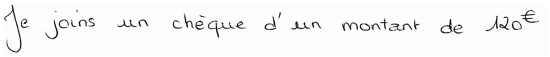
(k) Changement de luminosité



(l) Modification du contraste



(m) Flou aléatoire



(n) Augmentation de la netteté

FIGURE 5.7 – Illustration du résultat de différentes augmentations appliquées sur une image. L'image originale, issue du jeu de données de reconnaissance d'écriture moderne RIMES [Aug+06], est visible dans la figure (a). Certaines transformations sont volontairement exagérées pour illustrer les transformations.

variations dans les images, notamment sur la taille des caractères, tout en simulant des lignes de texte inclinées.

Augmentation de l'écriture D'autres transformations de l'image visent à déformer l'écriture plus spécifiquement. Certaines approches [Pui17 ; Mic+19 ; WZG21 ; CCP22 ; CCP23] proposent d'appliquer des transformations pour altérer l'épaisseur du tracé. De l'érosion affine l'épaisseur du trait, alors que de la dilatation est utilisée pour épaissir la taille du trait.

De plus, Simard et al. [SSP+03] proposent de déformer des caractères isolés de manière locale en appliquant des déformations élastiques aléatoires (aussi appelées distorsions de grille). Wigington et al. [Wig+17] appliquent quant à eux ces déformations élastiques aléatoires sur de l'écriture manuscrite. Cette transformation utilise une grille de points de contrôle qui sont déplacés aléatoirement afin d'obtenir des déformations locales des caractères. Ces déformations locales permettent de simuler des variations aléatoires de l'écriture réelle dues à de faibles contractions des muscles de la main [SSP+03], tout en conservant le style d'écriture de l'auteur. Cette transformation est particulièrement réaliste et est aujourd'hui considérée par une majorité d'approches [Str+18 ; Dut+18 ; Mic+19 ; YHM20 ; CCP22 ; Kan+22 ; WZG22 ; CCP23].

Transformation au niveau des pixels de l'image Enfin, d'autres transformations peuvent être réalisées en modifiant directement les valeurs des pixels des images. De manière classique, le contraste ainsi que la luminosité sont modifiés aléatoirement. Soullard et al. [Sou+19] ainsi que Singh et al. [SK21] ajoutent quant à eux du bruit gaussien pour altérer les images. Kang et al. [Kan+22] proposent de modifier la netteté de l'image pour rendre l'image en partie floue, ou pour accentuer la netteté de l'image. Cela permet ainsi de simuler des images de lignes de texte prisent en photo avec des réglages imprécis, qui peuvent être courants dans certains cadres d'application. Coquenet et al. [CCP22] proposent quant à eux de modifier aléatoirement la résolution de l'image, et ainsi de simuler des lignes de texte capturées avec des appareils de moins bonne qualité.

Applications de données augmentées

Les augmentations de données sont particulièrement utiles pour générer de la diversité dans les données d'entraînement. Elles permettent aux modèles existants de réduire le CER d'entre 5 et 25% relativement [Pui17 ; Dut+18 ; YHM20 ; SK21 ; CCP23] sur des jeux de données modernes comme IAM [MB02] et RIMES [Aug+06].

Augmentations en apprentissage Les augmentations de données peuvent être générées de plusieurs manières possibles. Une première manière consiste à générer lors d'une phase de pré-traitement un nombre fini de données augmentées, et de s'en servir en tant que jeu de données d'entraînement. Cela permet d'effectuer des calculs en dehors de la phase d'apprentissage et de potentiellement gagner du temps. En revanche, le nombre

de données augmentées reste limité et demande de l'espace pour sauvegarder les images. Cette première méthode est rarement utilisée [PMV16].

La seconde méthode consiste à augmenter des données de manière dynamique lors de l'entraînement. L'ensemble des paramètres d'augmentation sont alors obtenus aléatoirement à chaque *batch* de données traité lors de l'entraînement d'un réseau de neurones. Cette approche permet de générer une quantité virtuellement infinie de variations de données.

La seconde méthode est, à ce jour, utilisée par la majorité des approches [Pui17 ; Str+18 ; dAr+22] et possède bien plus d'avantages pour des réseaux de neurones, du fait de la plus grande variabilité proposée en entrée des réseaux.

Augmentations en test et réduction des erreurs de prédiction Les augmentations sont traditionnellement utilisées pour augmenter virtuellement la taille d'un jeu de données d'apprentissage. Additionnellement, certaines approches considèrent l'utilisation d'augmentation de données lors de la phase de prédiction [Blu+14 ; San+16 ; Str+18 ; Dut+18 ; Sou+19].

Une première solution consiste à utiliser des augmentations sur les image de test, de la même manière que pour des images lors de l'entraînement. L'approche proposée par ParisTech à l'occasion de la compétition sur le jeu de données READ 2018, utilise par exemple une telle approche [Str+18].

Une solution plus couramment utilisée consiste à générer plusieurs images augmentées aléatoirement à partir d'une image réelle. Des prédictions sont ensuite obtenues sur ces multiples images. La méthode proposée par OSU [Str+18] à l'occasion de la compétition READ 2018, utilise 38 images augmentées, avant de choisir la prédiction majoritaire obtenue sur ces images. Dutta et al [Dut+18] utilisent 25 images augmentées et proposent à la place de calculer une moyenne des activations de la dernière couche de leur modèle. Cette activation moyenne est utilisée pour obtenir une prédiction finale.

De la même manière, une combinaison des prédictions obtenues par plusieurs modèles peut être réalisée pour obtenir une réduction des résultats. L'algorithme ROVER introduit par Fiscus et al. [Fis97] propose ainsi d'utiliser plusieurs modèles entraînés, et de combiner les multiples prédictions obtenues au moyen d'une étape d'alignement des mots, suivie d'une étape de vote. Stuner et al. [SCP17] proposent plus tard une amélioration de l'algorithme, en allégeant la complexité de la procédure d'alignement en comptant le nombre d'espace (ou de mots) dans les prédictions, ainsi qu'en intégrant un vocabulaire dans le processus de vote pour favoriser des mots existants. La méthode ROVER est par

exemple utilisée par l'approche proposée par RWTH dans le cadre de la compétition sur le jeu de données READ 2016 ou encore dans des travaux d'A2iA [Blu+14], qui proposent de combiner 16 et 9 modèles respectivement.

Enfin, Soullard et al [Sou+19] proposent d'utiliser l'approche ROVER pour combiner des prédictions obtenues sur 20 images augmentées aléatoirement en test. Ils adaptent pour cela l'algorithme d'alignement au niveau des caractères, avant de choisir la prédiction la plus probable selon l'algorithme de vote, au niveau caractère.

Les augmentations en test apportent de meilleurs résultats de l'ordre de 5% relativement [Dut+18]. Cela permet en général de consolider la prédiction finale, qui aura tendance à réduire le nombre d'erreurs comparé à une prédiction isolée. Le modèle est alors moins sensible à de faibles variations dans les images. En revanche, des augmentations lors d'une phase de test demandent en contrepartie un temps proportionnel au nombre de fois où chaque image est augmentée.

Limites de l'augmentation de données

L'état de l'art des augmentations de données est riche en contenu, et a montré que ces augmentations apportent d'excellents résultats dans différents domaines, ainsi que celui de la reconnaissance d'écriture. Dans nos travaux, nous utiliserons ces différentes transformations pour apporter une variabilité accrue des images. Cependant, l'apport se limite à des données annotées existantes, et le contenu textuel appris reste donc limité aux labels du jeu d'apprentissage. Dans le cas d'architectures profondes qui demandent beaucoup de données annotées, comme des architectures Transformer, l'augmentation de données peut se révéler être une solution insuffisante, notamment en l'absence de nouveaux contenus textuels.

5.3.3 Génération de données synthétiques modernes

Le processus de génération de données synthétiques permet de générer des données annotées supplémentaires associées à de nouvelles transcriptions. Elles permettent, en complément avec des augmentations de données, de produire des images dans un style d'écriture plus ou moins varié selon le but, allant même jusqu'à être capable d'imiter le style d'écriture d'auteurs spécifiques. Comparé à de l'augmentation de données seule, la génération de données synthétiques offre ainsi une bien plus grande variabilité dans les données vues lors de l'entraînement.

Bien que cela bénéficie à l'ensemble des approches de reconnaissance d'écriture manuscrite, la génération de données synthétiques a été popularisée récemment avec l'arrivée d'architectures Transformers [SK21 ; Kan+22 ; WZG22 ; dAr+22 ; CCP23 ; Li+23]. La majorité des processus de génération de données concernant des approches basées sur des architectures de Transformer, cette section sera donc fortement axée autour de ces architectures.

Les architectures Transformers profitent particulièrement de l'apport de données synthétiques, permettant une réduction considérable du CER allant de 6% à 39% relativement, selon la complexité du réseau [Kan+22 ; SK21 ; WZG22 ; dAr+22]. Les architectures de Transformer larges nécessitant bien plus de données annotées pour s'entraîner, elle tirent alors grandement parti des données synthétiques. En comparaison, des architectures légères ont beaucoup moins besoin de données annotées. Les données synthétiques profitent tout de même à ces dernières, mais dans une plus faible mesure.

En effet, sur des documents modernes, 10 000 lignes de textes annotées (l'ordre de grandeur du jeu de données IAM [MB02] par exemple) se révèlent insuffisantes pour entraîner efficacement des architecture de Transformer larges [Kan+22]. Kang et al. [Kan+22] entraînent ainsi leur architectures sur 138 000 données synthétiques tandis que l'architecture TrOCR proposée par Li et al. [Li+23] est quant à elle entraînée sur des centaines de millions de lignes de textes générées à partir de fichiers PDF. De plus, les architectures Transformer profitent de l'ajout de nouvelles transcriptions pour apprendre à modéliser la langue.

Parmi les approches de génération de données synthétiques, deux approches que nous détaillerons se distinguent principalement :

— une approche de génération classique, basée sur l'utilisation de polices d'écriture et d'augmentations ;

— ainsi que des réseaux de neurones génératifs (GAN).

Processus de génération classique à base de polices d'écriture

L'approche classique pour générer des données synthétiques se décompose en général en trois étapes principales. Dans un premier temps, un texte à générer est sélectionné parmi une vaste liste de textes. Dans un second temps, une police d'écriture, imprimée ou manuscrite, est sélectionnée pour générer ce texte sous la forme d'une image de texte. Enfin, diverses transformations et dégradations sont appliquées à l'image pour la rendre réaliste et plus naturelle. Ce processus simple permet de générer des données synthétiques

réalistes se rapprochant d'écritures modernes [Kan+22 ; WZG22 ; dAr+22]. De plus, elles sont produites à un coût moindre comparé à l'annotation humaine de données réelles.

Contenus textuels Les données textuelles pour générer des données synthétiques peuvent provenir de différentes sources selon les approches. Le type de documents, la période et le nombre de données peuvent ainsi varier.

D'Arce et al. [dAr+22] utilisent le jeu de données textuel Brown [FK61] pour pré-entraîner leur modèle. Ce jeu de données est composé d'un million de mots provenant de diverses publications dans les années 1960. D'autres approches [SK21 ; Li+23] considèrent l'utilisation d'articles provenant de Wikipédia, avec donc des données textuelles plus récentes. En particulier, Singh et al. [SK21] s'entraîne avec le jeu de données WikiText-103 [Mer+16] regroupant 103 millions de mots (530 millions de caractères) issus de 28 595 articles Wikipédia.

Pour entraîner leur architecture large, Kang et al. [Kan+22] utilisent 138 000 lignes de textes générées à partir de divers e-books. Li et al. [Li+23] utilisent quant à eux une quantité de textes bien plus grande en comparaison pour entraîner leur architecture très large. Leur architecture est pré-entraînée dans un premier temps avec 684 millions de lignes de textes obtenues à partir de fichiers PDF. Dans un second temps, les auteurs utilisent le jeu de données IIIT-HWS [KJ16], ainsi que 53 000 reçus imprimés, pour entraîner efficacement leur architecture.

Polices d'écriture Pour apporter de la variabilité dans les formes à reconnaître, le choix des polices d'écriture, ainsi que le nombre de polices d'écriture ,jouent un rôle important. En particulier, le style de police d'écriture peut être pertinent ou non selon le type d'écritures réelles à reconnaître.

Pour des écritures imprimées, l'approche TrOCR proposée par Li et al. [Li+23] utilise des polices d'écriture imprimée pour générer des données synthétiques à partir de fichiers PDF. De plus, les auteurs utilisent deux polices d'écriture imprimée spécifiques pour générer des reçus réalistes. Singh et al. [SK21] ainsi que Coquenet et al. [CCP23] proposent d'utiliser des polices d'écriture imprimée pour apprendre la structure de paragraphes ou de pages. Même dans le but de reconnaître des écritures manuscrites, ils observent tout de même un gain important en utilisant des polices imprimées.

Cependant, les polices d'écriture imprimée sont visuellement très différentes du style des écritures manuscrites modernes. Ces dernières contiennent bien plus de complexité, comme par exemple diverses ligatures, ou encore des lignes contenant des ascendants et

des descendants se chevauchant. Ainsi, les approches abordant le problème de la reconnaissance d'écriture manuscrite moderne considèrent bien plus fréquemment l'utilisation de polices d'écriture manuscrite pour générer des données synthétiques. Tandis que Kang et al. [Kan+22] ainsi que d'Arce et al. [dAr+22] utilisent respectivement 387 et 400 polices d'écriture manuscrite, l'approche TrOCR [Li+23] considère quant à elle 5 427 polices d'écriture manuscrite. Ces différentes approches permettent de générer des images réalistes, et bien que ces polices d'écriture puissent certes différer du style des écritures réelles, les architectures profitent tout de même largement de l'apport de variabilité additionnelle.

Augmentations et dégradations L'utilisation de polices d'écriture permet ainsi de générer des images avec du contenu textuel. Cependant, une dernière étape très importante consiste à appliquer diverses augmentations et dégradations pour obtenir des écritures variées et réalistes. L'ensemble des augmentations présentées dans la section 5.3.2 peuvent ainsi être utilisées. De plus, Kang et al. [Kan+22], d'Arce et al. [dAr+22], ainsi que Singh et al [SK21] avec une approche au niveau page, ajoutent par exemple des fonds aléatoires en niveau de gris. L'utilisation de déformations élastiques est notamment très pertinente, car cette déformation permet de donner un côté réaliste dans les écritures en simulant des oscillations incontrôlées des muscles de la main, et donc de la variabilité intra-scripteur.

Réseaux de neurones génératifs

Des réseaux de neurones génératifs peuvent aussi être utilisés pour générer des données synthétiques. Ils permettent en particulier de générer du texte, tout en imitant un style d'écriture fourni au réseau. Ces approches utilisent des réseaux de générations adverses (*Generative Adversarial Networks* ou GAN) et permettent de générer des images de lignes de texte moderne réalistes [Fog+20 ; Dav+20 ; GW21]. Bhunia et al. [Bhu+21] utilisent quant à eux une approche basée sur une architecture de Transformer.

Ces approches génératives nécessitent l'utilisation d'un reconnaisseur de texte lors de leur entraînement [Fog+20 ; Dav+20 ; Vög+21 ; GW21 ; Bhu+21]. Celui-ci a pour but d'entraîner le réseau de neurones à générer du texte reconnaissable et en accord avec le texte demandé. Entraîner un réseau de neurones génératif pourra potentiellement donner de meilleurs résultats en comparaison à une approche basée sur des polices d'écriture et de l'augmentation, mais nécessitera en revanche des données annotées en quantité suffisante pour entraîner convenablement le reconnaisseur de texte, ainsi que l'approche générative.

Cette contrainte limite actuellement l'utilisation d'approches génératives pour fournir des données suffisantes à une architecture de reconnaissance de texte manuscrit.

5.3.4 Génération de données synthétiques anciennes

Difficulté de la génération pour des documents anciens Tandis que les approches de génération de données synthétiques permettent d'obtenir facilement des images de textes modernes, la tâche de génération de textes anciens réalistes est quant à elle bien plus complexe en comparaison. En effet, pour générer des données synthétiques réalistes au style des documents anciens, il est alors important de reproduire les nombreuses difficultés des documents anciens (évoquées dans la section 5.1.1), comme par exemple des dégradations ou des ligatures complexes.

Approches classiques Ces différentes contraintes limitent l'applicabilité des approches classiques utilisées pour générer des données synthétiques modernes. Ainsi, seules quelques approches permettent de générer des données synthétiques au style des documents anciens. Journet et al. proposent l'approche DocCreator [Jou+17] pour générer des documents complets dans un style moderne ou ancien. Leur approche permet d'appliquer diverses méthodes pour construire un document réaliste, ainsi que le dégrader, avec notamment de nombreuses transformations. Bien que leur approche soit plus pertinente pour transformer des documents réels, ils proposent de générer du texte à partir de caractères individuels segmentés de manière automatique. Cela reste cependant non applicable pour des écritures manuscrites contenant en particulier des ligatures.

Approches génératives Malgré la difficulté de la tâche, Vögtlin et al. [Vög+21] ainsi que Madi et al. [Mad+22] proposent des approches à base de GAN pour synthétiser des documents anciens. Ces approches utilisent toutes deux une fenêtre carrée de taille fixe comme base pour leur processus de génération. Cette fenêtre de longueur inférieure à celle d'une ligne est capable de contenir quelques mots situés sur quelques lignes consécutives.

L'approche de Vögtlin et al. [Vög+21] permet de générer des documents synthétiques anciens à partir de deux documents : un document contenant le texte à générer, et un autre document dont le style ancien sera imité. La fenêtre carrée est déplacée consécutivement sur les différentes parties de l'image et permet alors de transformer le document textuel en lui transférant le style ancien du document. Cependant, cette approche, de même que les approches génératives présentées dans la section 5.3.3, se base elle aussi

sur un reconnaisseur de texte, et demande donc des données annotées pour être entraînée convenablement. L'approche HST-GAN proposée par Madi et al. [Mad+22] se base à la place sur l'utilisation d'un squelette du tracé, obtenu automatiquement à partir d'une étape de binarisation. Un fond, ainsi que de l'encre autour du squelette du tracé sont ensuite générés pour obtenir des données synthétiques au style ancien. Contrairement à l'approche de Vögtlin et al., celle de Madi et al. ne nécessite pas de reconnaisseur de texte. En revanche, l'étape de binarisation nécessite un paramétrage en fonction du document.

Cependant, ces approches utilisent des fenêtres pour transférer consécutivement le style ancien. Or, sur des régions voisines, ces approches ne garantissent pas des résultats cohérents sur les bordures [Vög+21]. Dans le cadre de la génération de lignes synthétiques au style ancien, ces approches à base de fenêtre ne sont en conséquence pas adaptées.

5.3.5 Discussions

Comme nous l'avons vu dans la section 5.2.3, les architectures Transformer pourraient fortement contribuer à la reconnaissance d'écriture manuscrite ancienne. Cependant, ces architectures complexes dépendent fortement de nombreuses augmentations et de l'ajout de données synthétiques pour être entraînées efficacement.

Un processus de synthèse de données au style ancien à base de polices d'écriture manuscrite dans des styles anciens, d'augmentations et de déformations pourrait ainsi permettre de produire des données synthétiques à moindre coût. Nous pensons que de telles données synthétiques pourraient alors fortement aider l'entraînement d'architectures Transformer, grâce à l'apport de données annotées avec de nouveaux contenus textuels. À notre connaissance, un processus de synthèse d'écritures manuscrites au style ancien à base de polices d'écriture, n'a pas été proposé.

Cependant, la génération de données synthétiques au style des documents anciens est loin d'être un processus trivial, à cause d'écritures complexes, ainsi que de nombreuses dégradations. Il semble alors important pour un tel système de prendre en compte ces nombreux aspects afin de permettre l'entraînement d'architectures Transformer, ainsi que de bons résultats, sur des documents anciens.

Dans le chapitre 8, nous présenterons en tant que contribution de cette thèse un algorithme de génération de données synthétiques permettant de générer des données au style des documents modernes, mais aussi au style des documents anciens.

Deuxième partie

Contributions

Comme nous l'avons vu, les approches habituellement utilisées dans l'état de l'art peuvent être améliorées grâce à l'utilisation de couches de Transformer. Ces couches permettraient ainsi une meilleure modélisation d'informations contextuelles, ou encore un entraînement plus rapide. Cependant, les architectures Transformer sont particulièrement complexes à entraîner avec le peu de données disponibles dans le domaine de la reconnaissance d'écriture manuscrite. L'annotation de données manuscrites est en effet un processus coûteux, et c'est en particulier le cas pour des données anciennes.

Dans cette seconde partie, nous présentons ainsi les différentes contributions réalisées au cours de cette thèse. Elles sont axées autour de deux axes majeurs.

— D'une part, nous proposons des architectures de Transformer légères, conçues pour être efficaces à la fois sur des lignes de texte moderne, mais aussi sur des lignes de texte ancien beaucoup plus complexes. Elles ont notamment pour but de minimiser la complexité des entraînements ainsi que le besoin en données annotées.

— D'autre part, nous mettons en place diverses stratégies pour entraîner efficacement ces architectures et pour minimiser les taux d'erreur. L'accent est en particulier mis sur les documents anciens qui sont bien plus complexes.

Le chapitre 6 introduira l'architecture CFTNN en tant que première contribution de la thèse. Cette première architecture de Transformer est à la fois légère et permet une prédiction rapide. Elle s'inspire directement des architectures CRNN et utilise des blocs d'encodeur Transformer en remplacement des couches récurrentes. Elle a été proposée au cours de la première année de thèse, c'est-à-dire en 2021. Nous rappelons qu'il n'existait alors pas de travaux sur les architectures de Transformer dans le domaine.

Nous introduirons ensuite dans le chapitre 7, une architecture de Transformer autorégressive à la fois légère et modulable. En plus d'un encodeur Transformer, cette architecture inclut aussi un décodeur Transformer, ce qui lui permet d'apprendre une modélisation de la langue efficace pour réduire les taux d'erreur. Elle reste particulièrement légère, et elle est capable d'être déclinée en plusieurs variations pour s'adapter aux quantités de données disponibles. Cette contribution est plus récente, et s'est appuyée sur les récents travaux dans le domaine, au fur et à mesure de leur publication.

Dans le chapitre 8, nous proposons un algorithme pour générer des données synthétiques. Il peut être utilisé pour à la fois générer des données synthétiques au style des écritures modernes, ou encore dans le style des documents anciens. Ces données synthétiques peuvent alors être utilisées pour permettre un meilleur entraînement des architectures de Transformer légères proposées.

Enfin, au cours du chapitre 9, nous introduisons différentes stratégies visant à réduire les taux d'erreur obtenus par nos architectures. Une première stratégie d'entraînement et de spécialisation vise à permettre l'obtention de taux d'erreur faibles lorsque le nombre de données annotées est limité à quelques documents spécifiques. Enfin, une stratégie de prédiction basée sur de l'augmentation d'images en test est proposée pour permettre une réduction des taux d'erreur.

Chapitre 6

Réseau de neurones convolutifs à Transformer rapide pour la reconnaissance d'écriture manuscrite

---

Dans ce chapitre, nous proposons comme contribution, une première architecture de Transformer pour résoudre la tâche de reconnaissance de ligne d'écriture manuscrite : l'architecture convolutive à Transformer rapide, abrégée en CFTNN (*Convolutional Fast Transformer Neural Network*). L'architecture CFTNN est inspirée des architectures CRNN, dans lesquelles les couches récurrentes sont remplacées par des blocs d'encodeurs Transformer, qui permettent un processus de prédiction non-autorégressif.

De par l'utilisation d'un encodeur Transformer en remplacement des couches récurrentes, l'architecture CFTNN pourrait permettre les avantages suivants :

— une meilleure modélisation de l'information contextuelle, à la fois à des échelles locales et globales de la séquence, grâce à l'utilisation de blocs d'encodeur Transformer et du mécanisme d'attention induit, permettant en quelque sorte une généralisation des couches BLSTM ;

— une meilleure vitesse d'apprentissage et de prédiction, grâce à l'utilisation du mécanisme d'attention non-autorégressif et non-causal induit par un encodeur Transformer et sa forte parallélisation ;

— enfin, elle reste globalement très légère avec seulement 3,5M de poids entraînables, ce qui la rend adaptée à l'apprentissage sur un nombre restreint de données.

L'architecture CFTNN est donc particulièrement intéressante, puisqu'elle permettrait de bien meilleures capacités d'apprentissage en comparaison à une architecture CRNN classique. De plus, dans des scénarios où la vitesse de prédiction est particulièrement importante, elle peut permettre un décodage bien plus rapide.

Il est à noter que ce chapitre se base sur des travaux qui ont été réalisés au commencement de cette thèse (2020–2021). À notre connaissance, il n'existait donc pas à cette date, de travaux publiés concernant des architectures de Transformer pour la reconnaissance d'écriture. Durant cette période, Diaz et al. [Dia+21] ont évalués différentes architectures dans une étude non publiée, dont une architecture basée sur un encodeur Transformer qui est globalement similaire à l'architecture CFTNN proposée. Elle utilise en revanche plus de paramètres que la nôtre (environ 12M, soit quatre fois plus) et est entraînée avec bien plus de données annotées. De plus, l'architecture proposée par d'Arce et al. [dAr+22], possède aussi de nombreux points communs avec l'architecture CFTNN, mais a en revanche été publiée beaucoup plus récemment. Ce chapitre se base ainsi uniquement sur l'état de l'art présenté dans les chapitres 2 et 3.

Nous organisons le chapitre de la manière suivante : la section 6.1 présente l'architecture en détail ; nous aborderons ensuite dans la section 6.2 les aspects liés à son entraînement ; et enfin, nous proposons un bilan du chapitre dans la section 6.3.

6.1 Architecture convolutive à Transformer rapide

Dans cette section, nous fournissons une description détaillée de l'architecture convolutive à Transformer rapide (CFTNN) proposée. Notons aussi que l'architecture CFTNN constitue aussi la base de l'architecture Transformer que nous présenterons plus tard dans le chapitre 7, et que des résultats obtenus avec cette architecture seront présentés dans les chapitres 10 et 11.

La section 6.1.1 présentera une vue globale de l'architecture CFTNN, tandis que les sections 6.1.2 et 6.1.3 aborderont en détail les différentes parties qui la constituent.

6.1.1 Vue globale de l'architecture

L'architecture CFTNN reprend les principes des architectures CRNN, avec des blocs d'encodeur Transformer (introduits dans la section 3.3.1, p. 53) en remplacement des couches récurrentes. Une visualisation de l'architecture est disponible dans la figure 6.1.

L'architecture se divise en deux parties principales. Dans la première partie, la base convolutive (*convolutional backbone*) a pour but l'extraction automatique de caractéristiques à partir de l'image de ligne de texte. Cette partie sera abordée en détail dans la section 6.1.2. Dans la seconde partie, un encodeur Transformer est utilisé pour permettre

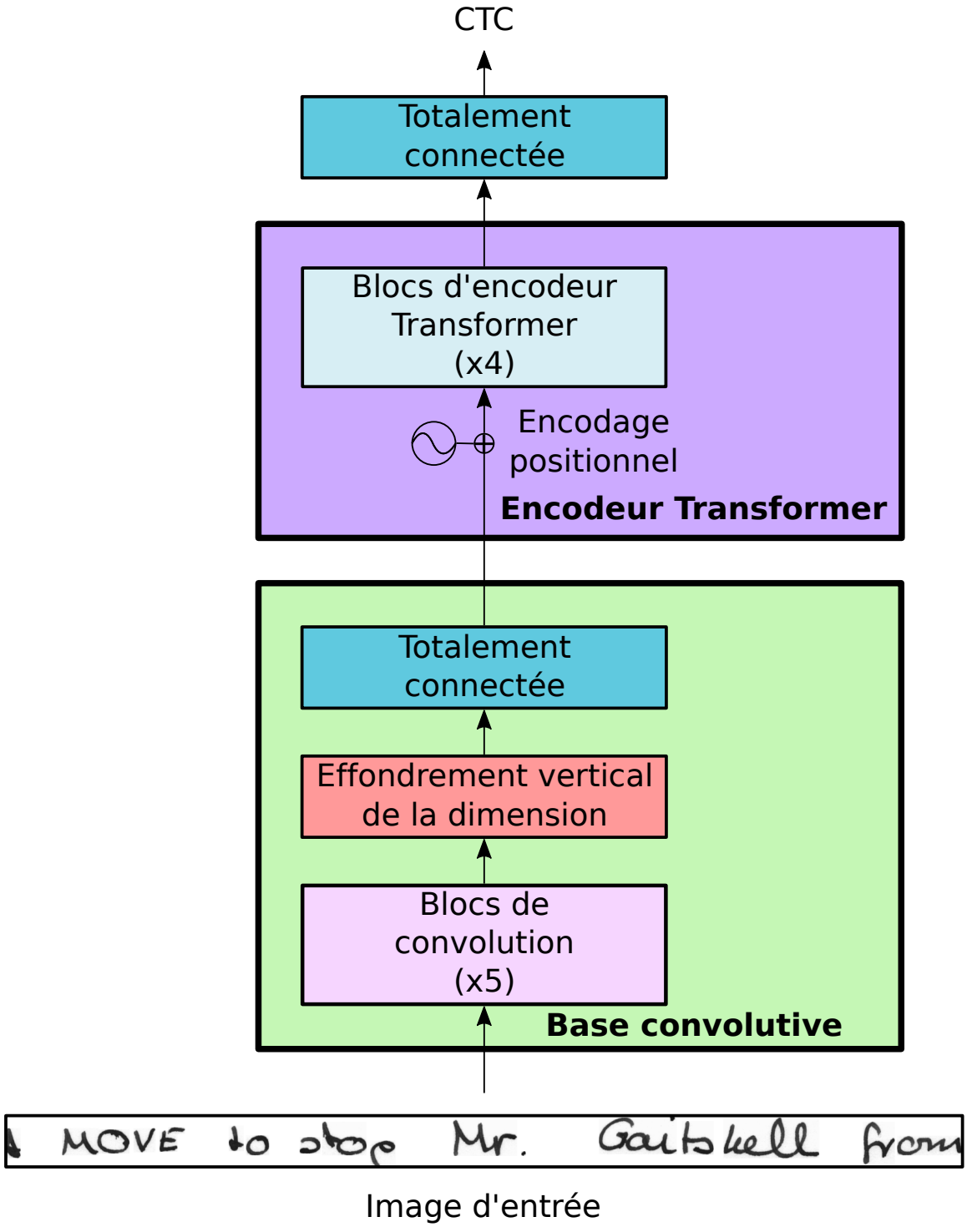


FIGURE 6.1 – Illustration de notre architecture CFTNN. Cette architecture se décompose en deux parties principales, qui reflètent le plan de la section : la base convolutive composée de couches de convolution ; et un encodeur Transformer utilisant plusieurs blocs d'encodeur Transformer.

l'intégration d'informations contextuelles issues de l'intégralité de la séquence. Plus de détails seront présentés au sein de la section 6.1.3.

Une couche totalement connectée est utilisée en sortie de l'encodeur pour produire une sortie sous la forme d'une séquence de scores (ou probabilité en utilisant un *softmax*) des caractères. Elle possède autant de neurones que le nombre de classes désirées, c'est-à-dire autant que le nombre de caractères dans l'alphabet considéré.

Notre architecture est globalement légère, avec 3,5M paramètres entraînables au total, dont 238k paramètres pour la base de convolution et 3,2M paramètres pour les blocs d'encodeur Transformer.

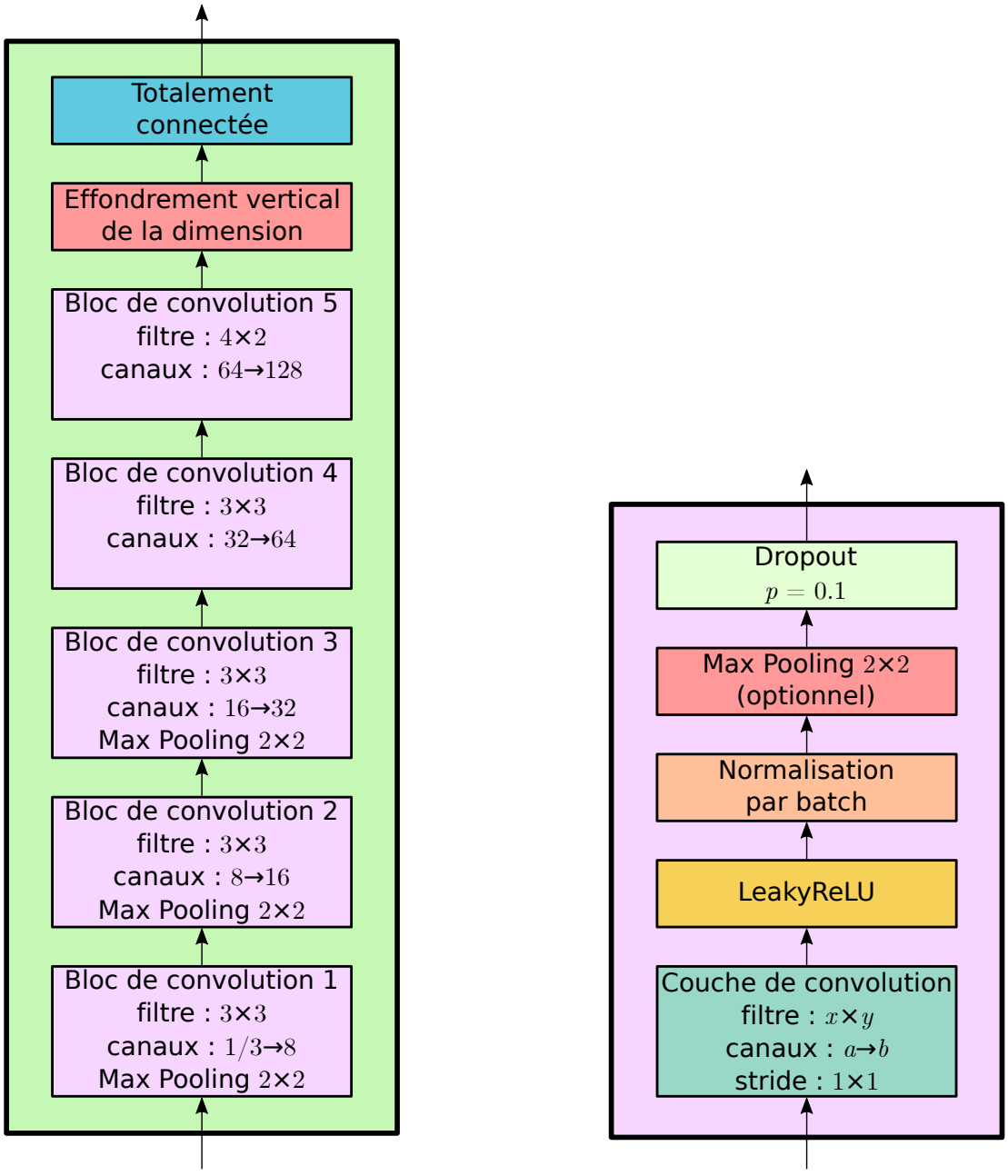
6.1.2 Base convolutive

La base convolutive (*convolutional backbone*) de l'architecture CFTNN permet de traiter une image de ligne de texte, et d'en extraire automatiquement un ensemble de caractéristiques, grâce à plusieurs blocs de convolution. Elle vise en particulier à produire une représentation riche sous la forme d'une séquence de vecteurs. Les caractéristiques produites contiennent en particulier de l'information locale et limitée à la taille du champ réceptif induit par les blocs de convolution consécutifs.

Blocs de convolution

La base convolutive de notre architecture est principalement composée de cinq blocs de convolution. Nous détaillons dans cette section les différents constituants de ces blocs, ainsi que leurs fonctions. La figure 6.2b montre un schéma de la composition de ces blocs de convolution tandis que la figure 6.2a illustre la base convolutive dans son ensemble, ainsi que les paramètres de chacun des blocs.

Couches de convolution Chaque bloc de convolution est composé d'une couche de convolution. Dans le but d'extraire une représentation riche, nous augmentons au fur et à mesure la dimension (c'est-à-dire le nombre de canaux) des cartes de caractéristiques intermédiaires. Ainsi, les couches de convolution produisent successivement des cartes de caractéristiques avec : 8, 16, 32, 64 et 128 caractéristiques. Cette augmentation croissante du nombre de caractéristiques est utilisée de manière classique par les différentes approches en vision par ordinateur [KSH12 ; SZ14 ; He+16], ainsi qu'en reconnaissance d'écriture [SBY16 ; SDN16 ; BM17 ; Pui17], et permet notamment de produire des représentations hiérarchiques de caractéristiques de plus en plus complexes. Les couches de



(a) Vue détaillée des différentes couches composant la base convolutive de notre architecture.

(b) Schéma détaillé d'un bloc de convolution.

FIGURE 6.2 – Illustration de la base convolutive de l'architecture CFTNN. La figure (a) montre une vue générale de la base convolutive, tandis que la figure (b) détaille un bloc de convolution.

convolution utilisent un filtre de convolution de taille 3×3, à l'exception de la cinquième couche qui utilise un filtre de taille 4×2. Ce dernier filtre au format vertical vise à suivre la forme des caractères. Ce même procédé est utilisé par quelques architectures CRNN pour améliorer la reconnaissance des caractères [SDN16 ; BM17]. Les filtres sont appliqués avec un déplacement (*stride*) d'un pixel que ce soit horizontalement ou verticalement, et nous n'utilisons pas de remplissage (*padding*) sur les bordures.

Fonction d'activation Après chaque couche de convolution, nous utilisons une fonction d'activation de type $LeakyReLU$ [MHN13]. Cette fonction d'activation permet d'ajouter des non-linéarités dans l'architecture, ce qui offre aux blocs plus d'expressivité. De plus, elle permet de filtrer les valeurs négatives, tout en conservant les valeurs positives. Il en résulte alors une représentation parcimonieuse, avec quelques valeurs significatives, ce qui permet d'accélérer l'apprentissage en le focalisant sur des informations jugées importantes.

Normalisations des activations Au sein des blocs de convolutions, nous utilisons de la normalisation à l'échelle d'un même lot (*batch*) d'images (*batch normalization*) [IS15]. Cette fonction de normalisation est utilisée en combinaison avec les couches convolutives par la majorité des approches, et a montré de bons résultats lorsque les tailles de *batch* utilisées à l'entraînement sont suffisamment importantes [He+16 ; Sze+16 ; Hua+17 ; Xie+17]. Elle permet de limiter la prépondérance de certaines caractéristiques, et donc de permettre un entraînement moins biaisé. De plus, l'utilisation de couches de normalisation permet une normalisation des valeurs des gradients, et ainsi un entraînement plus stable.

Pooling Dans les trois premiers blocs de convolution, nous utilisons une opération de *max-pooling* de taille 2×2, avec un déplacement de 2 pixels horizontalement et verticalement. Cette opération permet de réduire rapidement la largeur et la hauteur des cartes de caractéristiques intermédiaires, en conservant uniquement la valeur maximale d'une caractéristique dans une fenêtre de taille 2×2.

Dropout Dans nos blocs de convolution, nous utilisons du *dropout* spatial [Tom+15] avec une probabilité d'effacement de 10%. Le *dropout* spatial efface des cartes de caractéristiques entières, contrairement au *dropout* classique [Hin+12] qui efface des caractéristiques indépendantes aléatoirement. Il est donc plus adapté pour des tâches liées aux images. Cet effacement est particulièrement utile, puisqu'il force les réseaux de neurones à éviter l'utilisation de seulement quelques caractéristiques, et donc à apprendre diverses

caractéristiques importantes, robustes et générales. Ainsi, le réseau est moins sujet à du surapprentissage et à des valeurs de gradients très importantes. Cette pratique est utilisée par la majorité des approches de classification d'images [KSH12 ; SZ14 ; He+16 ; Hua+17] ou en reconnaissance d'écriture [Pha+14].

Effondrement vertical de la dimension

Comme pour les couches récurrentes dans un modèle CRNN traditionnel, les blocs de Transformer nécessitent l'utilisation d'une représentation séquentielle. Pour passer d'une représentation des caractéristiques en deux dimensions (des cartes de caractéristiques) vers une organisation séquentielle des caractéristiques, un "*pooling*" vertical est utilisé. Celui-ci a pour but de résumer l'ensemble des vecteurs de caractéristiques d'une colonne en un seul vecteur de caractéristiques.

À la suite des blocs de convolution, nous utilisons ainsi un "*pooling*" vertical basé sur une couche de convolution, une fonction d'activation *LeakyReLU*, ainsi qu'une normalisation par *batch*. Comparé à un *pooling* classique basé sur l'utilisation d'un *max-pooling* ou d'un *average-pooling*, une couche de convolution apprend automatiquement la manière de résumer une colonne de caractéristiques.

Couche totalement connectée

Enfin, nous ajoutons une couche totalement connectée avant l'encodeur Transformer, pour passer du nombre d'activation des couches de convolution (128) à celui des blocs Transformer ($D = 256$).

6.1.3 Encodeur Transformer

Un encodeur Transformer, composé de plusieurs blocs d'encodeurs Transformer, est utilisé dans la seconde partie de l'architecture CFTNN. Tandis que les couches de convolution permettent de modéliser un contexte local, restreint par la taille du champ réceptif, les blocs de Transformer permettent l'intégration d'informations contextuelles sur l'ensemble de la séquence de caractéristiques. Cela permet ainsi de produire une séquence de caractéristiques consolidées grâce à l'intégration de ces informations contextuelles.

Nous utilisons quatre blocs d'encodeur Transformer dans l'architecture CFTNN. Ces blocs sont expliqués au sein de la section 3.3.1 (p. 53) et illustrés dans la figure 3.5a (p. 54). Chaque bloc d'encodeur Transformer est composé d'une couche d'auto-attention

à têtes multiples, avec quatre têtes d'attention de 64 neurones chacune. Une couche de *dropout* [Hin+12] avec une probabilité d'effacement de 0,2 est appliquée au sein de ces blocs pour limiter le surapprentissage, et nous utilisons de la normalisation par couche (*layer normalization*) [BKH16], ce qui a pour effet de stabiliser l'entraînement.

De plus, nous ajoutons de l'information positionnelle sur la séquence d'entrée du décodeur Transformer en utilisant un encodage positionnel. Cet ajout est en particulier important pour le mécanisme d'attention inhérent aux architectures Transformer, qui n'effectue pas de traitement séquentiel, et qui n'a donc pas, de base, accès à l'information de l'ordre des caractéristiques de la séquence. L'encodage positionnel est appliqué tel que décrit dans la section 3.1.3 (p. 46).

Comparées à des couches récurrentes, les couches d'auto-attention à têtes multiples permettent une meilleure intégration de l'information contextuelle à court et long terme. Ce mécanisme d'attention offre des capacités similaires, voire supérieures, à celles des couches de BLSTM (introduites dans la section 2.2.1, p. 33), et propose en quelque sorte une généralisation de ces dernières.

L'encodeur Transformer permet un traitement de la séquence plus rapide que celui effectué par des couches récurrentes grâce à la parallélisation des calculs. De plus, comme nous utilisons un mécanisme d'attention non-causal, notre architecture de Transformer est alors non-autorégressive, ce qui permet des gains en vitesse considérables. En effet, il n'est pas nécessaire d'effectuer un traitement séquentiel de la séquence de caractéristiques, ce qui permet de gagner en temps d'apprentissage, ainsi qu'en vitesse de prédiction. Cela se révèle particulièrement utile dans des scénarios où une vitesse de prédiction importante est nécessaire, ou encore lorsque des volumes importants de données sont à traiter.

6.2 Détails de l'entraînement

Dans cette section, nous apportons quelques détails sur l'entraînement de l'architecture CFTNN. La section 6.2.1 abordera la fonction de coût utilisée, tandis que nous parlerons des paramètres de l'entraînement dans la section 6.2.2.

6.2.1 Fonction de coût

L'architecture CFTNN est basée sur le principe d'un découpage en trames, comme expliqué dans la section 1.3 (p. 22). Ce découpage est réalisé par la base convolutive de l'architecture, avec comme taille de trame, la taille du champ réceptif, soit une largeur

d'au plus 46 pixels. Pour une image en entrée dimensionnée à une hauteur de 128 pixels, cela correspond à environ un tiers de la hauteur. Enfin, chaque trame consécutive est placée avec un décalage relatif de 8 pixels en raison de l'utilisation de *pooling*.

Le nombre de trames (et donc la taille de la séquence produite par notre architecture) étant bien plus grand que le nombre de caractères dans la vérité terrain, nous utilisons une approche de classification temporelle connexioniste (CTC) pour l'entraînement.

Pour obtenir la séquence prédite en inférence, nous utilisons un décodage glouton (*greedy decoding*, aussi appelé *best-path decoding*), en choisissant le caractère le plus probable pour chaque trame. Les doublons de caractères consécutifs similaires, ainsi que les jetons spéciaux "`Blank`" sont ensuite supprimés, comme expliqué dans la section 2.2.4 (p. 36).

6.2.2 Paramètres de l'entraînement

L'architecture CFTNN utilisant globalement peu de poids, elle peut donc être entraînée avec des tailles de *batch* importantes et peu de ressources. Elle est typiquement entraînée avec une taille de *batch* de 64 images. Au sein d'un même *batch*, les images d'entrées sont redimensionnées à une hauteur de 128 pixels tout en conservant le ratio. Cependant, les images n'ont pas forcément la même longueur et du *padding* est ajouté sur les images plus petites que l'image la plus longue. Nous utilisons alors un masque dans les couches d'attention à têtes multiples pour s'assurer que l'attention n'utilise pas des valeurs en dehors de la séquence.

Plus de détails sur le processus d'apprentissage de l'architecture seront fournis plus tard, dans la section 10.1.4 (p. 156).

6.3 Avantages de l'architecture CFTNN

Comme nous l'avons vu, l'architecture CFTNN permet de nombreux avantages, qui lui offrent une plus grande capacité en comparaison à un CRNN traditionnel.

D'une part, son utilisation de blocs d'encodeur Transformer (combiné à un encodage positionnel) en remplacement des couches récurrentes, peut permettre une bien meilleure intégration d'informations contextuelles à court et long terme. Cela viendrait en quelque sorte généraliser les couches BLSTM traditionnellement utilisées dans des architectures CRNN.

De plus, l'architecture CFTNN étant basée sur des couches d'attention non-causales, elle est à la fois rapide lors du processus d'entraînement, ainsi qu'en prédiction. Elle est donc particulièrement adaptée dans des scénarios où la vitesse de prédiction est importante.

Avec seulement 3,5M de paramètres, l'architecture CFTNN reste très légère en comparaison à d'autres architectures de Transformer, que ce soit en traitement automatique de la langue, ou en reconnaissance d'écriture. Elle s'entraîne ainsi avec un nombre relativement limité de données annotées. Elle nous semble donc parfaitement adaptée pour des tâches de reconnaissance d'écriture sur des documents modernes ou sur des documents anciens.

Compte tenu des divers avantages de l'architecture CFTNN, nous pensons que cela en fait une contribution originale. Nous rappelons en particulier qu'il n'existait pas à notre connaissance, et à cette date, de travaux concernant des architectures Transformer pour la reconnaissance d'écriture. Nous avons par ailleurs présenté les travaux relatifs à ce chapitre lors du *Doctoral Consortium de l'ICDAR 2021* [Bar+21b], sous la forme d'un poster.

Afin de valider l'architecture CFTNN, nous présenterons les résultats relatifs à celle-ci dans les chapitres 10 et 11.

Enfin, nous pensons que l'ajout d'un décodeur Transformer directement intégré à l'architecture pourrait permettre de meilleures capacités, comme pourrait le permettre un modèle de langue externe. Nous en avons en particulier discuté dans la section 4.3.1 (p. 67). Cela pourrait ainsi permettre de meilleurs résultats, et en particulier pour la reconnaissance de documents anciens. Cette constatation motive notamment les travaux réalisés dans le chapitre suivant.

Chapitre 7

Architecture de Transformer autorégressive légère et modulable pour la reconnaissance d'écriture manuscrite

---

Dans ce chapitre, nous proposons une seconde architecture de Transformer pour la reconnaissance d'écriture manuscrite : l'architecture de Transformer autorégressive légère et modulable, abrégée en MLT (*Modular Lightweight Transformer*).

L'architecture MLT intègre en particulier un décodeur Transformer, qui réalise alors un procédé autorégressif pour traiter la séquence et apporter des capacités de modélisation de la langue. Elle combine en particulier, des capacités de reconnaissance optique des caractères, réalisées par une base convolutive et un encodeur Transformer, avec des capacités de modélisation de langue obtenues avec un décodeur Transformer. L'architecture est donc entraînable de bout en bout.

En opposition aux architectures de Transformer larges [Kan+22 ; Li+23], nous choisissons de conserver un nombre relativement peu important de poids entraînables. L'architecture MLT est donc légère et capable de s'entraîner avec peu de données annotées. De plus, elle peut facilement être modulée dans sa légèreté pour être entraînée sur les scénarios avec très peu de données annotées. Elle est donc tout particulièrement adaptée pour la reconnaissance de documents anciens. À notre connaissance, c'est la première architecture de Transformer appliquée à la reconnaissance de lignes de textes sur des documents anciens et complexes.

Ces travaux ont été réalisés plus récemment, sur les deuxièmes et troisièmes années de thèse (de fin 2021 à début 2023). Ils prennent notamment en compte les récents travaux de l'état de l'art au fur et à mesure de leur publication.

Le chapitre est organisé comme suit. La section 7.1 aborde dans les détails l'architecture ainsi que les couches la composant. Dans la section 7.2, nous discutons ensuite de la modularité de notre architecture, et de comment elle peut être adaptée aux quantités de données annotées disponibles, et notamment les plus limitées. En outre, nous proposons deux variantes de l'architecture MLT. Enfin, la section 7.3 présente les différents détails nécessaires à la reproduction des entraînements de l'architecture.

7.1 Architecture de Transformer autorégressive légère

Dans cette section, nous présentons l'architecture MLT. En particulier, nous détaillons les différents constituants de l'architecture, dans le but qu'elle soit facilement reproductible. Notons que nous mettons à disposition de manière ouverte le code relatif à l'architecture présentée dans ce chapitre [1]. Nous discuterons cependant de son nombre relativement restreint de poids entraînables plus tard, dans la section 7.2.

7.1.1 Vue d'ensemble de l'architecture

Notre architecture MLT est composée de deux parties principales.

— Un encodeur basé sur l'architecture CFTNN, présentée dans le chapitre 6. Celui-ci, à la manière d'un modèle optique, vise à produire une séquence de caractéristiques riche à partir d'une image de ligne de texte. La section 7.1.2 détaillera cette partie, ainsi que diverses adaptations de l'encodeur.

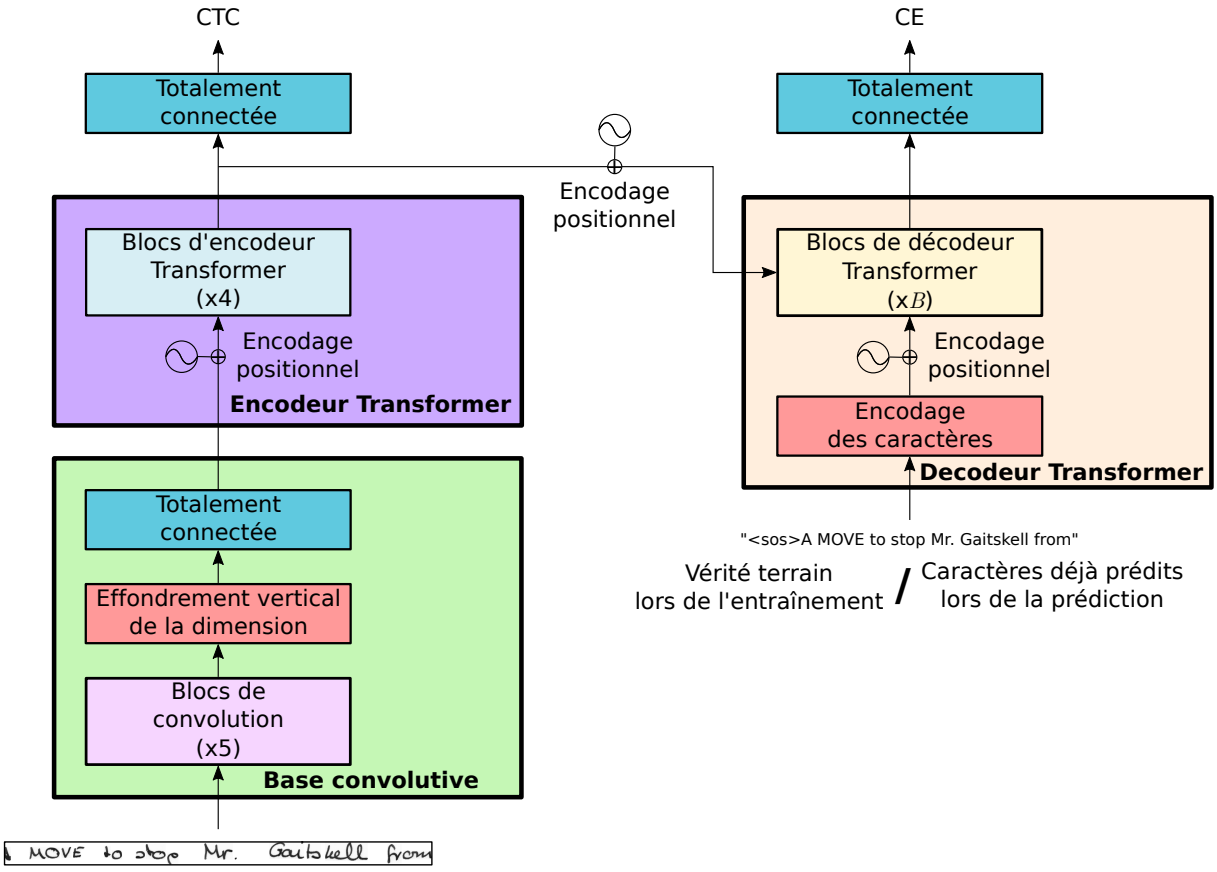
— Un décodeur utilisant des blocs de décodeur Transformer, dans le but de modéliser la langue et de produire une séquence avec moins d'erreurs. Cette seconde partie sera abordée dans la section 7.1.3.

Un schéma résumant l'architecture MLT est présenté dans la figure 7.1.

Tandis que les approches classiques de reconnaissance d'écriture considèrent traditionnellement un modèle optique et un modèle de langue entraînés séparément, l'architecture MLT permet de réaliser les deux fonctions à la fois et peut être entraînée de bout en bout.

Lors de l'étape d'inférence, l'architecture MLT suit un fonctionnement classique d'architectures encodeur-décodeur autorégressives. En une seule passe, l'encodeur extrait de

1. Le code est disponible à l'adresse suivante : `https://gitlab.inria.fr/intuidoc-public/vlt`



(a) Partie encodeur. (b) Partie décodeur.

FIGURE 7.1 – Schéma de l'architecture MLT. L'encodeur est illustré sur la partie gauche de l'image (a), tandis que le décodeur l'est sur la partie droite (b).

l'information de l'image de ligne de texte. Le décodeur effectue quant à lui un traitement séquentiel en plusieurs passes, pour produire les caractères. À chaque étape, il utilise ainsi la séquence des caractères déjà produits, ainsi que l'information produite par l'encodeur.

Lors de l'étape d'apprentissage, le processus séquentiel du décodeur est cependant effectué en une unique étape. En effet, nous utilisons de l'apprentissage forcé avec la vérité terrain, ainsi qu'un masque pour les couches d'attention causales. Nous aborderons ce sujet en détail dans la section 7.3.2.

Nous entraînons l'architecture MLT en utilisant à la fois les sorties de l'encodeur et du décodeur. Pour ce faire, nous utilisons une combinaison de deux fonctions de coût, que nous détaillerons plus tard dans ce chapitre (section 7.3.1). Comme pour l'architecture

précédente, une couche de neurones totalement connectée est utilisée pour produire les scores (ou probabilité après application d'un *softmax*) de chaque caractère dans l'alphabet.

7.1.2 Partie encodeur

La partie encodeur de l'architecture MLT (figure 7.1a) agit comme un modèle optique. L'encodeur a pour but de produire une séquence de caractéristiques informative sur les caractères de la ligne de texte, à partir d'une image de ligne de texte. L'encodeur de l'architecture MLT est quasiment identique à l'architecture CFTNN dans son ensemble (section 6.1, p. 98). Il utilise ainsi une base convolutive, ainsi que des couches d'encodeur Transformer.

L'architecture MLT inclut plus de poids en comparaison à l'architecture CFTNN de par l'ajout d'un décodeur. Pour entraîner l'architecture MLT, nous réduisons alors les tailles de *batch* utilisées. De ce fait, nous remplaçons les normalisations par *batch*, par de la normalisation par couche (*layer normalization*) [BKH16]. Cette dernière permet un entraînement plus efficace et de meilleurs résultats sur des tailles de *batch* réduites.

La sortie de l'encodeur Transformer (et donc des quatre blocs d'encodeur Transformer utilisés) sert deux utilisations dans l'architecture. D'une part, elle est passée en paramètre d'une couche totalement connectée pour produire une séquence de scores de caractères. La séquence de scores est ensuite utilisée pour entraîner l'architecture avec une approche de classification temporelle connexioniste (CTC) [Gra+06]. D'autre part, la sortie de l'encodeur Transformer est utilisée en tant qu'entrée des différents blocs de décodeur Transformer. Cela permet ainsi à l'encodeur de produire une séquence de caractéristiques contenant de l'information sur les caractères identifiés, tout en conservant la même dimension des caractéristiques entre l'encodeur et le décodeur ($D = 256$).

7.1.3 Partie décodeur

La partie décodeur de l'architecture MLT (figure 7.1b), peut quant à elle être vue comme un modèle de langue intégré directement à l'architecture. En effet, elle est entraîné à produire le caractère suivant, et agit donc comme un modèle de langue au niveau caractère. D'autre part, nous avons vu dans la section 4.3.1 (p. 67) qu'un décodeur Transformer semble en effet agir de la sorte. Cette partie a donc pour but d'améliorer la séquence de caractéristiques produites par l'encodeur, grâce à l'intégration d'informations contextuelles supplémentaires.

Pour ce faire, la partie décodeur de l'architecture MLT est composée d'un empilement de B blocs de décodeur Transformer, comme présentés dans la section 3.3.2 du chapitre 3 (p. 55). Le nombre B de blocs de décodeur Transformer permet de moduler la légèreté de l'architecture, et donc de s'adapter à différentes quantités de données annotées. Nous en reparlerons dans la section 7.2.

Dans cette section, nous détaillons les principes des différentes couches composant le décodeur de l'architecture MLT.

Séquences en entrée du décodeur

Tout comme expliqué dans la section 3.3.2 (p. 55), le décodeur de l'architecture MLT prend en entrée deux séquences. Il utilise la séquence des caractères déjà prédits, mais aussi la séquence de caractéristiques produite par l'encodeur.

La séquence des caractères déjà prédits (ou alors la vérité terrain à l'entraînement comme nous utilisons de l'apprentissage forcé) est tout d'abord encodée en une représentation vectorielle, comme expliqué dans la section 3.1.2 (p. 45). Pour cela, nous utilisons un encodage *one-hot* des caractères, suivi d'une couche de neurones totalement connectés pour obtenir une représentation informative avec D caractéristiques. Nous ajoutons ensuite de l'information positionnelle en utilisant un encodage positionnel sinusoïdal.

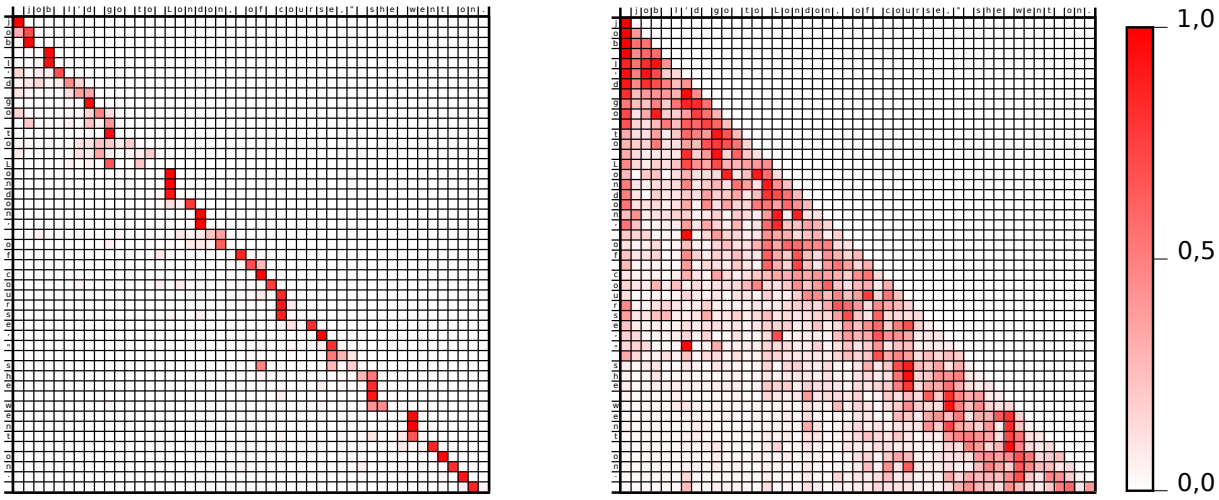
D'autre part, la séquence de caractéristiques produite par l'encodeur est utilisée dans les couches d'attention mutuelle des blocs décodeurs. Cette séquence riche en information sur les caractères identifiés dans l'image, sert en particulier à prédire le caractère suivant. Nous ajoutons à cette séquence de l'information positionnelle via un encodage positionnel sinusoïdal supplémentaire. Bien que cette séquence contienne théoriquement de l'information positionnelle provenant de l'encodage positionnel placé avant les couches d'encodeur Transformer, nous avons observé de meilleurs résultats en ajoutant cette information positionnelle supplémentaire. Cela pourrait notamment permettre de simplifier l'apprentissage des couches utilisées dans le décodeur, avec de l'information positionnelle disponible moins profondément.

Auto-attention pour modéliser la langue

Chaque bloc de décodeur Transformer utilise en premier lieu de l'auto-attention à têtes multiples. Cette couche a intuitivement pour but de prédire le caractère à venir le plus probable, étant donné la séquence de caractéristiques associée aux caractères déjà prédits.

Cette couche d'auto-attention permet ainsi de réaliser une modélisation de la langue au niveau des caractères.

Chaque couche d'auto-attention utilise $D = 256$ neurones, et est composée de 4 têtes d'attention. Chaque tête d'attention est ainsi capable de modéliser la langue avec des contextes plus ou moins lointains. Ce principe est notamment illustré par la figure 7.2.



(a) Tête d'attention modélisant de l'information contextuelle locale.

(b) Tête d'attention modélisant de l'information contextuelle lointaine.

FIGURE 7.2 – Poids obtenus par différentes têtes d'auto-attention d'une même couche d'attention à têtes multiples. Chaque ligne correspond à un caractère à prédire. Les zones rouges montrent sur quels caractères se portent l'attention pour prédire le caractère suivant. Dans cet exemple, la figure (a) montre une modélisation d'un contexte proche par une tête d'attention, tandis que la figure (b) montre la modélisation du contexte lointain effectuée par une autre tête.

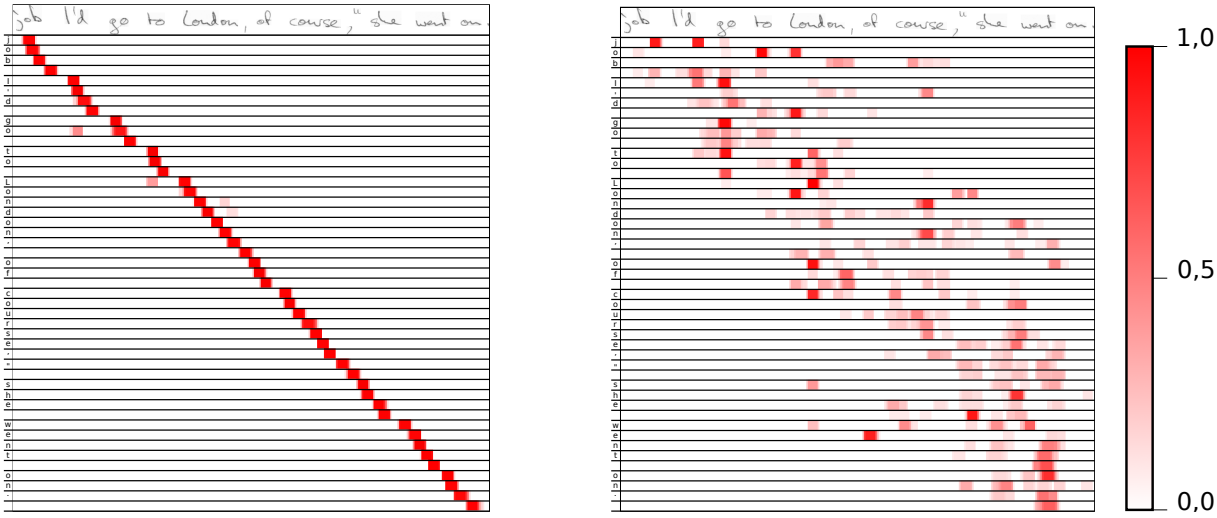
Enfin, comme pour les autres couches d'attention à têtes multiples, nous utilisons du *dropout* avec une probabilité d'effacement de 20%, ainsi que de la normalisation par couche [BKH16].

Mécanisme d'attention image-séquence

À la suite de l'auto-attention causale, chaque bloc de décodeur Transformer utilise de l'attention mutuelle (ou encodeur-décodeur). Dans cette couche d'attention mutuelle, le vecteur de requête $\boldsymbol{Q}$ dérive directement des caractéristiques produites par la couche d'auto-attention précédente. Les vecteurs de clés $\boldsymbol{K}$ et de valeurs $\boldsymbol{V}$ sont quant à eux obtenus par la séquence de caractéristiques provenant de l'encodeur.

L'attention mutuelle a pour but l'intégration d'informations visuelles sur l'image de la ligne de texte, dans le but de prédire des caractères. Par exemple, il pourra être intéressant

d'étudier la présence d'espaces, de formes verticales souvent associées à des caractères ascendants ou ascendants, ou encore de diverses formes. La figure 7.3 montre la réponse de deux couches d'attention mutuelle.



(a) Poids d'attention mutuelle avec une progression linéaire.

(b) Tête d'attention fortement focalisée autour de certains caractères.

FIGURE 7.3 – Poids d'attention obtenus pour des couches d'attention mutuelle. Chaque ligne correspond à un caractère à prédire. Pour chaque caractère à prédire, l'attention se porte sur différentes parties de l'image (ou pour être plus exact : de la séquence de caractéristiques produite par l'encodeur sur l'image d'entrée). Ici, les poids d'attention de deux têtes d'attention sont illustrés, avec pour la figure (a) une attention avec une progression linéaire dans l'image, et dans la figure (b) une tête d'attention principalement concentrée sur des formes précises.

De même, chaque couche d'attention mutuelle est composée de $D = 256$ neurones, de 4 têtes d'attention, et utilise du *dropout* avec une probabilité d'effacement de 20% ainsi que de la normalisation par couche [BKH16].

7.2 Modularité de l'architecture

Les blocs de décodeur Transformer représentent une grande partie (environ 50%) des poids de l'architecture, et peuvent facilement surapprendre la langue. L'utilisation d'un nombre de blocs trop important peut alors conduire à de moins bonnes performances. Il convient alors d'utiliser un nombre de blocs de décodeur en accord avec le nombre de données annotées disponibles.

Nous proposons ainsi deux variantes de notre Architecture de Transformer autorégressive légère en tant que contributions de la thèse.

7.2.1 LT : Architecture de Transformer légère

L'architecture de Transformer légère, ou LT (*Lightweight Transformer*) utilise $B = 4$ blocs de décodeurs Transformer, et contient un total de 7,7M de paramètres. Elle reste légère par rapport aux autres architectures de Transformer autorégressives dans le domaine de la reconnaissance d'écriture, qui peuvent atteindre jusqu'à des centaines de millions de paramètres [SK21 ; WZG21 ; Kan+22 ; WZG22 ; Li+23].

7.2.2 VLT : Architecture de Transformer très légère

Dans le but de limiter le surapprentissage de la langue lorsque le nombre de données annotées est très limité, nous présentons une seconde variante plus légère. Nous proposons ainsi l'architecture de Transformer très légère, ou VLT (*Very Lightweight Transformer*), qui utilise seulement $B = 2$ blocs de décodeur Transformer dans le but d'apprendre à modéliser la langue avec peu de données annotées. Elle comptabilise quant à elle 5,6M de paramètres entraînables.

L'architecture VLT nous semble ainsi adaptée aux documents anciens, qui contiennent en général des quantités très limitées de données annotées, tout en bénéficiant de l'apport de capacités de modélisation de la langue.

7.3 Détails de l'entraînement

Dans cette section, nous présentons le processus d'apprentissage de nos architectures, dans le but de permettre une reproduction des entraînements réalisés. Plus de détails seront fournis plus tard, dans la section 10.1.4 (p. 156).

Les hyper-paramètres des architectures proposées sont globalement similaires à ceux utilisés pour l'architecture CFTNN (présentés dans la section 6.2, p. 104). Nos architectures LT et VLT utilisant cependant plus de poids, nous utilisons une taille de *batch* de 32. De plus, nous utilisons des fonctions de coût différentes, ainsi que de l'apprentissage forcé pour entraîner le décodeur.

La section 7.3.1 abordera ainsi la combinaison des deux fonctions de coût utilisées. Dans la section 7.3.2, nous détaillerons le processus d'apprentissage de l'encodeur avec de l'entraînement forcé, ainsi que la manière d'obtenir une prédiction.

7.3.1 Fonction de coût hybride

Michael et al. [Mic+19] ont proposé l'utilisation d'une fonction de coût hybride pour permettre un meilleur entraînement des architectures récurrentes. Dans le but de simplifier l'entraînement des architectures Transformer, nous utilisons aussi une combinaison de fonctions de coût. Nous combinons à la fois une approche de classification temporelle connexioniste (CTC) [Gra+06] pour entraîner l'encodeur, ainsi qu'une fonction de coût d'entropie croisée (CE pour *Cross-Entropy*) pour entraîner le décodeur.

Notre fonction de coût total $\mathcal{L}$, est obtenue de la manière suivante :

$$\mathcal{L} = \lambda \cdot \mathcal{L}_{CTC} + (1 - \lambda) \cdot \mathcal{L}_{CE} \tag{7.1}$$

avec λ le poids relatif entre les deux fonctions de coût $\mathcal{L}_{CTC}$ et $\mathcal{L}_{CE}$. $\mathcal{L}_{CTC}$ est définie de la manière suivante :

$$\mathcal{L}_{CTC}(\boldsymbol{y}, \hat{\boldsymbol{y}}^{\boldsymbol{enc}}) = -log\Big(\sum_{\pi \in \mathcal{B}^{-1}(\boldsymbol{y})} p(\pi|\hat{\boldsymbol{y}}^{\boldsymbol{enc}})\Big) \tag{7.2}$$

avec $\hat{\boldsymbol{y}}^{\boldsymbol{enc}}$, la prédiction de l'encodeur, et $\mathcal{B}^{-1}(\boldsymbol{y})$, l'ensemble des alignements possibles produisant la séquence $\boldsymbol{y}$ en enlevant les répétitions de caractères et les labels spéciaux "`Blank`". La fonction de coût $\mathcal{L}_{CE}$ est quant à elle définie de la manière suivante :

$$\mathcal{L}_{CE}(\boldsymbol{y}, \hat{\boldsymbol{y}}^{\boldsymbol{dec}}) = -\sum_{i=1}^{N} log(\hat{\boldsymbol{y}}^{\boldsymbol{dec}}_{\boldsymbol{i},\boldsymbol{y_i}}) \tag{7.3}$$

avec $\hat{\boldsymbol{y}}^{\boldsymbol{dec}}$ la prédiction du décodeur, vue comme une séquence de probabilité de caractères. $\hat{\boldsymbol{y}}^{\boldsymbol{dec}}_{\boldsymbol{i},\boldsymbol{y_i}}$ correspond donc à la probabilité du label $\boldsymbol{y_i}$ à la i-ème position de la séquence de sortie du décodeur. Enfin, N, correspond à la longueur de la séquence.

Nous utilisons ces deux fonctions de coût dans le but de faciliter l'entraînement. De manière analogue aux couches résiduelles [He+16], cela offre un raccourci au gradient dans les couches les plus profondes (c'est-à-dire les premières couches de l'encodeur). L'encodeur profite alors du gradient provenant d'erreurs commises à la fois par la sortie du décodeur, mais aussi par la sortie de l'encodeur. L'encodeur est alors moins sujet à des phénomènes de disparition du gradient, tout en étant capable de converger plus rapidement.

De fait, l'entièreté de l'architecture est impactée, avec une potentielle amélioration des performances de reconnaissance de l'architecture, ainsi qu'une meilleure convergence, notamment avec moins de données annotées.

Dans la suite de ce manuscrit, nous utilisons une valeur de $\lambda = 0,5$ comme les deux fonctions de coût sont de même ordre de grandeur.

7.3.2 Apprentissage forcé et décodage en prédiction

Apprentissage forcé lors de la phase d'apprentissage

Le processus de décodage d'un décodeur Transformer nécessite un temps important à cause d'une prédiction étape par étape.

Pour accélérer ce processus lors de l'étape d'apprentissage, nous utilisons de l'apprentissage forcé [FL90] avec une probabilité de 1,0, comme proposé par Vaswani et al. [Vas+17]. Cela permet un apprentissage du processus de prédiction des caractères parallélisé, et ne dépendant pas de l'aspect séquentiel des prédictions.

De manière classique, un décodeur a pour but de prédire les différents caractères de la vérité terrain, ainsi qu'un jeton spécial "`<end of sequence>`" pour indiquer la fin du processus de prédiction. Cependant, dans le cadre d'apprentissage forcé, la vérité terrain est aussi utilisée en entrée du décodeur. Elle est décalée d'un caractère vers la droite (dans le sens de lecture), tout en ajoutant un jeton spécial "`<start of sequence>`" en début de séquence. Combiné avec un masque dans les couches d'auto-attention pour forcer la causalité du mécanisme d'attention, cela assure que le décodeur n'accède pas à des caractères futurs. Pour la prédiction d'un caractère, le décodeur a en revanche accès à l'intégralité des caractère précédents de la vérité terrain, c'est-à-dire la meilleure prédiction possible.

L'ensemble de ce processus d'apprentissage forcé permet alors de simuler en parallèle une prédiction des caractères étape par étape. Cela permet ainsi de fortement accélérer le processus d'apprentissage.

Phase d'inférence

Lors de la phase d'inférence, nous utilisons cependant un processus de décodage autorégressif, et donc étape par étape. À chaque étape de prédiction, le caractère avec la probabilité la plus élevée est prédit. La prédiction de la séquence est arrêtée lorsqu'un caractère de fin de séquence ("`<end of sequence>`") est prédit, ou lorsque la taille maximale de la séquence, que nous fixons à 150, est atteinte.

7.4 Avantage et limites de l'architecture

Dans ce chapitre, nous avons proposé une architecture de Transformer autorégressive, légère et modulable : l'architecture MLT. Elle combine un encodeur pour faire de la reconnaissance optique avec un décodeur pour modéliser la langue, tout en étant entraînable de bout en bout. Cette capacité de modélisation de la langue est utile pour corriger d'éventuelles erreurs, et en particulier pour la reconnaissance d'écritures anciennes [San+16 ; Str+18]. Dans le but de s'adapter aux quantités de données disponibles, nous proposons notamment deux variantes d'architectures : LT et VLT. Ces variantes sont obtenues en modulant le nombre de blocs de décodeur Transformer et donc la légèreté de l'architecture.

Architectures légères Nos architectures restent légères, avec un nombre de paramètres compris entre 5 et 10M. C'est nettement inférieur aux autres architectures du domaine qui peuvent utiliser jusqu'à 100M [Kan+22], voire 500M de paramètres [Li+23]. Cette différence s'explique notamment par une base convolutive bien plus légère (voir la section 6.1.2, p. 100), mais aussi des couches Transformer avec 256 neurones par couche contre 1 024 pour les architectures les plus larges [Kan+22 ; Li+23]. La légèreté des architectures proposées offre en conséquence de nombreux avantages pour la tâche de reconnaissance d'écriture, qui sont discutés dans les paragraphes suivants.

Entraînement à moindre coût En comparaison à des architectures bien plus larges, notre architecture peut être entraînée bien plus vite, et avec moins de ressources. Elle est donc relativement peu coûteuse, et ce gain n'implique pas nécessairement une baisse des performances.

Entraînement efficace avec peu de données annotées En comparaison à d'autres architectures de Transformer, nos architectures sont relativement légères, et donc capables de s'entraîner efficacement sur peu de données annotées. Cela fait donc de nos architectures, et en particulier VLT, des architectures adaptées à la reconnaissance d'écriture manuscrite ancienne. Nous verrons en particulier dans le chapitre 11 que nos architectures sont capables d'obtenir des résultats au niveau de l'état de l'art sur des documents anciens. En revanche, dans des scénarios avec beaucoup de données annotées (que nous n'étudions pas dans ce manuscrit), la légèreté de nos architectures peut limiter les performances obtenues.

Apprentissage à partir de zéro L'utilisation de modèles pré-entraînés peut être très intéressante pour obtenir de bonnes performances [Li+23]. En revanche, le réentraînement de ces modèles pour la tâche de reconnaissance d'écriture est loin d'être trivial à cause d'un manque de données annotées. Nos architectures, en raison de leur nombre restreint de poids, peuvent quant à elle être entraînées à partir de zéro.

Les travaux relatifs à l'architecture LT sur des documents modernes ont été publiés et présentés au workshop international DAS 2022 [Bar+22b]. Les travaux de ce même article ont par la suite été présentés au workshop national RFIAP 2022 [Bar+22a]. Des résultats sur des documents anciens de l'architecture LT, ont été présentés au symposium francophone SIFED 2022 [Bar+22c]. L'architecture VLT ainsi que des résultats sur les documents anciens ont été présentés à SIFED 2023 [Bar+23a], et un article de journal soumis dans la revue internationale IJDAR est en cours de relecture [Bar+23b].

Bien que nos architectures soient capables d'être entraînées efficacement sur peu de données annotées, elles profiteraient tout de même de données annotées supplémentaires. En particulier, l'ajout de nouveau contenu textuel permettrait d'améliorer les capacités de modélisation de la langue de nos architectures. C'est pourquoi nous nous penchons sur l'utilisation de données synthétiques dans le chapitre suivant.

Chapitre 8

Génération de données synthétiques

Dans le domaine de la reconnaissance d'écriture manuscrite, les architectures Transformer ont généralement besoin de données annotées supplémentaires pour obtenir de bons résultats. Ce besoin en données est fortement lié à la complexité et au nombre de poids des modèles. Pour compenser le manque de données annotées, les approches Transformer utilisent pour la plupart des données synthétiques [Kan+22 ; SK21 ; WZG22 ; dAr+22 ; CCP23 ; Li+23].

Les architectures proposées dans ce manuscrit sont bien plus légères en comparaison avec les autres architectures du domaine [Kan+22 ; Li+23]. Nous montrerons qu'elles peuvent être entraînées efficacement avec les données limitées d'un jeu de données et obtenir des résultats satisfaisants. Cependant, l'apport de données synthétiques bénéficierait tout de même à nos architectures légères en apportant une plus grande variété dans les styles d'écriture, ainsi qu'un contenu textuel accru. De plus, dans le cas de documents anciens, qui sont relativement complexes et peu annotés, l'utilisation de données synthétiques pour entraîner nos architectures semble bien plus pertinent. Cela aiderait en particulier là où le réentraînement de modèles pré-entraînés est loin d'être un processus évident face à la complexité de ce type de données.

Pour ces raisons, nous proposons d'entraîner nos architectures avec des données synthétiques au style des documents modernes ou anciens, en fonction du jeu de données étudié.

Dans ce chapitre, nous détaillons deux contributions principales de la thèse :

— un algorithme de génération de lignes de texte synthétiques dans le style des documents modernes (section 8.1) ;

— un algorithme de génération de données synthétiques dans le style visuel des documents anciens complexes, capable de générer aussi bien des lignes de texte que des paragraphes complets. À notre connaissance, une telle approche de génération

n'a pas été proposée pour entraîner des architectures à la reconnaissance d'écriture ancienne (section 8.2).

Nos deux algorithmes de génération de données synthétiques se basent sur l'utilisation de polices d'écritures manuscrites ainsi que diverses augmentations et dégradations. Ceci nous permet de générer des données synthétiques réalistes et à moindre coût.

La section 8.1 explique la manière dont nous procédons pour générer des lignes de texte synthétiques dans le style des documents modernes. Dans la section 8.2, nous présentons notre algorithme de génération de données synthétiques dans le style des documents anciens, adapté du précédent algorithme. Enfin, dans la section 8.3, nous présentons différents résultats visuels obtenus grâce à nos approches, tout en fournissant une analyse qualitative des images obtenues.

Notons que nous visons à fournir une explication des processus de génération la plus détaillée possible. Là où la majorité des approches existantes fournissent souvent peu de détails, nous avons pour objectif de rendre notre approche reproductible. De plus, nous mettons à disposition le code de nos algorithmes de génération de lignes synthétiques [1].

8.1 Génération de lignes synthétiques pour des documents modernes

Dans cette section, nous présentons la première contribution : notre algorithme de génération de lignes de textes synthétiques au style visuel des documents modernes. Nous nous concentrons en particulier sur la génération de lignes isolées dans le style des jeux de données IAM et RIMES, présentés dans la section 2.1 (p. 29). En particulier, nous ne considérons pas ici l'utilisation d'enjambements. La section 8.1.1 présente le processus de génération dans sa globalité, puis, nous présentons, dans la section 8.1.2 le contenu textuel utilisé, et dans la section 8.1.3 les polices d'écritures utilisées. Enfin la section 8.1.4 détaille les augmentations utilisées pour générer des données synthétiques modernes.

8.1.1 Description du processus de génération

Notre algorithme de génération de lignes de textes synthétiques au style des documents modernes se base sur trois étapes principales : la sélection d'un contenu textuel ; la génération de l'image de texte en utilisant une police d'écriture manuscrite ; ainsi qu'une

1. `https://gitlab.inria.fr/intuidoc-public/synthetic-handwriting-generation`

déformation aléatoire de l'image et de l'écriture. La figure 8.1 fournit une illustration de ce processus complet.

FIGURE 8.1 – Illustration du processus de génération de lignes synthétiques modernes en trois étapes.

Notre méthode a pour but de s'adapter facilement sur des types de données différents. Nous utilisons ainsi des fichiers de configuration pour paramétrer le choix des polices d'écritures, ainsi que pour paramétrer l'utilisation des diverses augmentations et dégradations.

8.1.2 Contenu textuel utilisé

Le contenu textuel joue un rôle important dans la génération de données synthétiques. Il permet d'apporter un nouveau vocabulaire, ainsi que de nouvelles tournures de phrases, pour entraîner les réseaux de neurones. Cela semble d'autant plus important pour des architectures Transformer qui sont capables de modéliser la langue. Notons que nous proposons de générer du texte dans les langages des jeux de données que nous étudions (*c.f.* sections 2.1 p. 29, et 5.1.2 p. 74).

Articles Wikipédia

Pour générer des données synthétiques modernes, nous utilisons du texte issu d'articles provenant de Wikipédia [2]. Pour les langues anglaise, française et allemande, nous utilisons uniquement une partition des nombreux articles disponibles. Cela nous permet d'avoir un contenu textuel à la fois suffisamment large et varié. Pour les langues danoises et suédoises, nous utilisons l'ensemble des articles disponibles, comme leur nombre est relativement faible. Un résumé des statistiques sur le contenu textuel de Wikipédia utilisé est disponible dans la table 8.1.

TABLE 8.1 – Statistiques sur les articles de Wikipédia utilisés pour générer des données synthétiques. Les chiffres sont donnés par langue, et après une phase de pré-traitement pour ne conserver que le contenu textuel exploitable.

	# articles	# paragraphes	# mots
Anglais	21 350	609 865	39 966 190
Français	107 686	1 521 062	72 308 412
Allemand	106 637	1 919 584	89 077 018
Danois	199	801	21 679
Suédois	294	1 081	44 039

Nous utilisons uniquement les articles écrits dans les mêmes langues que celles du jeu de données considéré. De plus, nous respectons la proportion des langues présentes dans un jeu d'entraînement, en pondérant les différentes langues. Par exemple, pour traiter le jeu de données IAM [MB02], nous utilisons uniquement les articles de Wikipédia écrits en anglais.

Pour utiliser les données provenant d'articles Wikipédia, nous procédons à un pré-traitement. Celui-ci permet de ne conserver que le contenu textuel présent dans un article, en enlevant les balises de mise en forme ; les titres de sections ; les traductions et étymologie de termes précis ; les tableaux ; les équations ; les algorithmes ; les listes ; les adresses web ; les références ; ainsi que les notes et commentaires.

Jeu de données de reconnaissance d'écriture

Nous utilisons aussi la vérité terrain sur la partition d'entraînement du jeu de données que l'on cherche à reconnaître pour obtenir du contenu textuel. Nous utilisons pour cela des

2. `https://dumps.wikimedia.org/backup-index.html`

vérités terrains au niveau paragraphe. Dans le cas où la vérité terrain au niveau paragraphe n'est pas disponible, nous concaténons les vérités terrains des lignes consécutives.

Bien que cela n'apporte pas de vocabulaire additionnel, en utilisant la vérité terrain au niveau paragraphe, nous générons de nouvelles lignes de texte avec du contenu textuel appartenant à des lignes consécutives. Cela permet ainsi d'apporter plus de variété à moindre coût lors de l'apprentissage.

Sélection du contenu textuel à générer

Pour générer une ligne de texte synthétique, nous commençons par sélectionner de manière uniforme un type de texte à générer parmi les différents types de textes disponibles (articles Wikipédia, vérité terrain du jeu de données, *etc*). Nous choisissons un paragraphe aléatoire dans l'ensemble du contenu textuel disponible offert par ce type de données (par exemple dans un des articles de Wikipédia).

Nous sélectionnons ensuite un mot au hasard dans ce paragraphe pour débuter notre ligne synthétique. Puis, nous choisissons un nombre de caractères dans la séquence aléatoire, en suivant une loi normale reprenant la moyenne et l'écart-type observés sur le jeu de données à reconnaître. Dans le cas où la séquence serait coupée au milieu d'un mot, nous ajoutons la fin de ce mot dans le texte à générer. Enfin, dans le cas où la fin d'un paragraphe serait atteinte, nous conservons un texte plus court, ce qui permet aussi de simuler des lignes de petite taille.

8.1.3 Polices d'écritures manuscrites modernes

Une fois le texte sélectionné, une police d'écriture aléatoire est utilisée pour obtenir une image de ligne de texte. Nous utilisons des polices d'écriture au style d'écritures manuscrites modernes [3]. Les polices d'écritures sont sélectionnées manuellement en fonction de leur style visuel pour imiter au mieux les différentes écritures du jeu de données. Dans le cas du jeu de données IAM [MB02] par exemple, nous utilisons un total de 32 polices d'écriture au style similaire à celles des écritures du jeu de données.

Notre algorithme de génération de lignes synthétiques accepte aussi bien les polices d'écriture au format TTF ou OTF. Le format de polices d'écriture OTF (pour *Open Type Font*) offre la possibilité d'activer différentes caractéristiques pour obtenir des variations

3. Les polices utilisées sont disponibles sur `https://fonts.google.com`, `https://www.dafont.com` et `https://www.p22.com`.

de caractères [4]. Ces caractéristiques permettent par exemple l'activation de ligatures entres certains caractères, résultant ainsi en des variation stylistiques de l'écriture générée. Un exemple des variations obtenues avec l'activation de ces caractéristiques est illustré à la figure 8.2.

(a) Image générée sans caractéristique activée.

(b) Résultat avec la caractéristique "`liga`" (ligatures) activée.

(c) Résultat avec la caractéristique "`salt`" (style alternatif) activée.

FIGURE 8.2 – Exemple d'images de lignes de texte obtenues en activant différentes caractéristiques de la police d'écriture "Professor". Les différences sont entourées en rouge pour plus de clarté. Ici, aucune transformation n'est appliquée aux images.

Pour ces polices au format OTF, nous activons aléatoirement chaque caractéristique disponible avec une probabilité de 30%. Parmi l'ensemble des polices d'écritures, nous accordons une probabilité (ou un poids) plus élevée d'être sélectionnée aux polices qui ont plus de caractéristiques implémentées [5]. En effet, celles-ci pourront donner lieu à plus de variétés en comparaison à d'autres polices. Le poids d'une police est ainsi augmenté de 30% de la valeur de base par caractéristique disponible.

8.1.4 Augmentations de l'image

Enfin, au cours d'une troisième étape, nous appliquons diverses transformations aléatoires sur l'image générée, de manière analogue à une augmentation de données. Cela permet d'ajouter de la diversité sur l'image ainsi que de rendre l'image bien plus réaliste.

4. Une liste des caractéristiques existantes est disponible sur `https://learn.microsoft.com/en-us/typography/opentype/spec/featurelist`.

5. Plus de détails sur la liste des polices, les poids et fichiers de configuration utilisés sont disponible à l'adresse suivante : `https://gitlab.inria.fr/intuidoc-public/synthetic-handwriting-generation`.

Nous appliquons des transformations parmi celles décrites ci-dessous. Chaque transformation est appliquée aléatoirement avec une probabilité de 30%, indépendamment des autres transformations [6].

Pour une représentation visuelle des transformations appliquées, nous invitons les lecteurs à se référer à la figure 5.7 (p. 83).

Érosion et dilatation du tracé De l'érosion et de la dilatation sont appliquées aléatoirement sur le tracé de l'écriture, ce qui permet d'affiner ou de grossir l'épaisseur du tracé. Lorsque cette augmentation est sélectionnée, l'érosion est appliquée dans 50% des cas, et la dilatation dans les autres cas. Pour éroder ou dilater le tracé, un filtre 2D est choisi aléatoirement parmi l'un des filtres suivants :

$$\begin{pmatrix} 0 & 1 & 0 \\ 1 & 1 & 1 \\ 0 & 1 & 0 \end{pmatrix}, \begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix}, \begin{pmatrix} 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{pmatrix}, \begin{pmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{pmatrix}, \begin{pmatrix} 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \end{pmatrix}$$

La nouvelle valeur d'un pixel est déterminée par la valeur maximale, ou minimale des pixels dans la zone de ce filtre. Une valeur de 1 dans le filtre indique qu'un pixel est pris en compte dans le calcul de la valeur d'un nouveau pixel, alors qu'une valeur de 0 indique qu'il est ignoré. La forme du filtre permet en particulier de déterminer à quel point le tracé est érodé ou dilaté.

Inclinaison de l'écriture Un changement aléatoire de l'inclinaison des caractères (donnant un style d'écriture italique) est appliqué sur l'image de ligne de texte. Pour cela, nous appliquons un cisaillement de l'image d'une valeur allant de 10° dans le sens anti-horaire, jusqu'à 40° dans le sens horaire. Lors de la transformation, les zones de l'image manquantes sont complétées par la couleur du fond, c'est-à-dire uniquement de la transparence, ou un fond blanc uni. Il est à noter que l'inclinaison est appliquée sur des écritures avec une inclinaison neutre. Si une police génère de base une écriture italique, nous la ramenons

6. Les fichiers de configuration relatifs aux augmentations sont disponible à l'adresse suivante : `https://gitlab.inria.fr/intuidoc-public/synthetic-handwriting-generation`.

à une inclinaison neutre. Pour ce faire, nous avons ajusté manuellement l'inclinaison de chacune des polices.

Déformations élastiques Nous appliquons des déformations élastiques pour simuler des variations dues aux muscles de la main [SSP+03], et permettre un rendu bien plus réaliste. Contrairement aux autres transformations, nous appliquons systématiquement une déformation élastique pour obtenir des lignes de texte synthétiques plus réalistes. Le code utilisé est une version adaptée de la transformation proposée par Coquenet et al. [CCP22] ainsi que Yousef et al. [YB20]. Pour réaliser cette transformation, nous utilisons une grille de contrôle de forme carrée, avec pour longueur de chacun des carrés $l = \frac{H}{5}$, où H représente la hauteur de l'image. Nous étirons ensuite l'image en déplaçant aléatoirement les points de contrôles définis par cette grille, d'une distance d'au plus $0,2 \times l$ dans chacune des directions, horizontales ou verticales. De plus, pour éviter une déformation trop brutale qui pourrait conduire à des caractères fortement étirés ou tassés, nous limitons la distance horizontale et verticale entre deux points de contrôles adjacents à des valeurs comprises dans l'intervalle $[0,8 \times l; 1,2 \times l]$. La nouvelle valeur des pixels est calculée en réalisant une interpolation sur les pixels déplacés. Une illustration de cette transformation est disponible dans la figure 8.3.

(a) Image synthétiques de base.

(b) Image synthétique déformée par une déformation élastique

FIGURE 8.3 – Illustration du processus de déformation élastique sur une image de ligne de texte synthétique. Sur ces images, la grille de contrôle utilisée ainsi que sa déformation est mise en valeur. Ici, nous montrons un exemple avec 3 carrés sur la hauteur de l'image au lieu de 5 pour illustrer la déformation avec plus de clarté.

Rotation de l'image Nous appliquons une rotation aléatoire d'au maximum 1° dans le sens horaire ou anti-horaire, ce qui permet de donner un côté légèrement oblique à la ligne de texte. De même, l'image est complétée par la couleur du fond, et une interpolation des pixels est réalisée pour calculer les nouvelles valeurs.

Transformation de la perspective Une modification aléatoire de la perspective de l'image permet de simuler différents angles de vue sur le texte. Nous appliquons alors une

transformation similaire à la déformation élastique présentée ci-dessus. Nous utilisons les quatre coins de l'image en tant que points de contrôle, et chaque coin est déplacé d'une valeur aléatoire comprise dans les intervalles $[-0,25 \times H; 0,25 \times H]$ verticalement et $[-0,25 \times W; 0,25 \times W]$ horizontalement, avec H et W la hauteur et la largeur de l'image respectivement.

Ajout d'un bruit gaussien Un bruit gaussien aléatoire est ajouté sur l'image, pour changer aléatoirement les valeurs des pixels. Nous utilisons un bruit gaussien avec une moyenne de 0 et un écart type de 5 (pour des valeurs de pixels dans $[\![0; 255]\!]$).

Modification des valeurs de l'image Enfin, nous utilisons un ensemble de trois transformations appliquées indépendamment les unes des autres (toujours avec une probabilité de 30%). Ces trois transformations modifient aléatoirement : la luminosité ; le contraste ; ainsi que la netteté de l'image. Chacune des transformations affecte ainsi positivement ou négativement la caractéristique en question d'au plus 25%.

8.1.5 Intégration à l'entraînement

Notre algorithme de génération permet d'obtenir des images de lignes de texte isolées, c'est-à-dire dans un format similaire aux données d'entraînements des différents jeux de données existants. Cela nous permet donc de facilement inclure ces images synthétiques, avec leur vérité terrain associée, dans des systèmes de reconnaissance d'écriture manuscrite existants.

Pour entraîner nos architectures, nous utilisons un mélange de données annotées réelles et de données synthétiques, avec la même proportion de données réelles que de données synthétiques. Chaque époque d'entraînement contient l'intégralité des N images de lignes de texte provenant d'un jeu de données existant, auxquelles sont ajoutées N images de lignes de textes synthétiques. Ces données synthétiques sont générées à la volée lors de l'entraînement, résultant ainsi en de nouvelles données synthétiques à chaque époque. Cela permet ainsi d'entraîner ces architectures avec une grande variabilité.

8.2 Génération de données synthétiques dans des styles anciens

Comme expliqué dans la section 5.1.1 (p. 72) de ce manuscrit, les documents anciens sont bien plus difficiles à reconnaître en comparaison aux documents modernes, en raison de nombreuses dégradations, d'un manque de données annotées, ainsi que d'écritures complexes.

Dans cette section, nous proposons un algorithme pour générer des données synthétiques dans le style des documents anciens. En particulier le cœur de l'approche consiste à générer des paragraphes complets. Cet algorithme se base sur la génération de données synthétiques pour des documents modernes, et nous présentons dans la section 8.2.1 comment ce dernier a été adapté afin de générer des données synthétiques au style des documents anciens. Dans la section 8.2.2, nous présentons le contenu textuel utilisé. La section 8.2.3 parle quant à elle des polices d'écritures utilisées pour générer des écritures manuscrites au style ancien. Enfin, la section 8.2.4 détaille les différentes augmentations et dégradations spécifiques utilisées pour donner un style ancien et réaliste à ces données synthétiques.

8.2.1 Adaptation du processus de génération à des paragraphes de documents anciens

Nous suivons le principe de la génération avec des polices d'écritures, que nous adaptons spécifiquement à des documents anciens.

Ces documents contiennent très fréquemment des enjambements d'une ligne sur l'autre, laissant apparaître sur une ligne des parties de caractères descendants de la ligne supérieure, ou ascendants de la ligne inférieure. De plus, avec des lignes inclinées ou mal segmentées, il n'est pas rare de voir des morceaux d'autres lignes sur une image de ligne de texte.

Nous proposons alors, à la place de la génération de lignes de textes isolées, de générer des paragraphes complets d'écritures manuscrites. Ainsi, la présence d'enjambements sur une ligne de texte est reproduite en générant plusieurs lignes proches. De plus, en utilisant des transformations aléatoires comme de la rotation, nous reproduisons des apparitions partielle d'autres lignes sur l'image d'une ligne.

Cette nouvelle approche se divise en cinq étapes que nous détaillons.

1. Nous sélections le contenu textuel d'un paragraphe complet choisi au hasard (à la place d'un texte court dans le cas de la génération de données synthétiques modernes).

2. Celui-ci est généré en utilisant des polices d'écritures manuscrites, et en ajoutant des retours à la ligne dans le texte.

3. Nous appliquons des augmentations pour déformer l'écriture générée, et la rendre plus réaliste.

4. Un fond aléatoire est ajouté à l'image, et nous appliquons diverses dégradations spécifiques aux documents anciens pour obtenir une image de paragraphe.

5. Nous procédons à l'extraction de lignes de textes isolées.

Ces différentes étapes sont illustrées à la figure 8.4. Cette approche peut aussi être considérée pour générer des lignes synthétiques modernes avec des enjambements.

Lors du processus de génération d'un paragraphe, un ensemble de paramètres permet de contrôler l'aspect visuel du paragraphe. La longueur maximale des lignes d'un paragraphe est choisie aléatoirement, pour une valeur allant jusqu'à doubler la valeur minimale possible (entre 100% et 200% d'une longueur fixée en paramètre). L'espace interligne, c'est-à-dire l'espace séparant chaque ligne, et contrôlant la présence d'enjambement, est choisi aléatoirement entre 50% et 100% de la hauteur maximale d'une ligne de texte. Enfin, nous ajoutons un faible décalage de la position de chaque ligne, horizontal, comme vertical, allant jusqu'à 5% de la hauteur maximale d'une ligne de texte.

Pour l'extraction de lignes de texte isolées depuis l'image de paragraphe, nous utilisons des boites englobantes. Lors de la génération du contenu textuel avec des polices d'écriture, la position initiale de ces boites englobantes est connue. Par la suite, lorsque nous appliquons différentes augmentations sur l'écriture, les coordonnées des boites englobantes sont mises à jour en fonction des paramètres des différentes augmentations utilisées. Il est cependant à noter que lors de certaines augmentations, nous omettons volontairement l'étape de mise à jour des coordonnées des boites englobantes, afin de simuler un placement aléatoire du texte au sein de la boite englobante. Cela concerne le faible décalage horizontal et vertical de la ligne de texte ainsi que des déformations élastiques. La figure 8.5 donne un exemple de la position des boites englobantes dans un paragraphe complet.

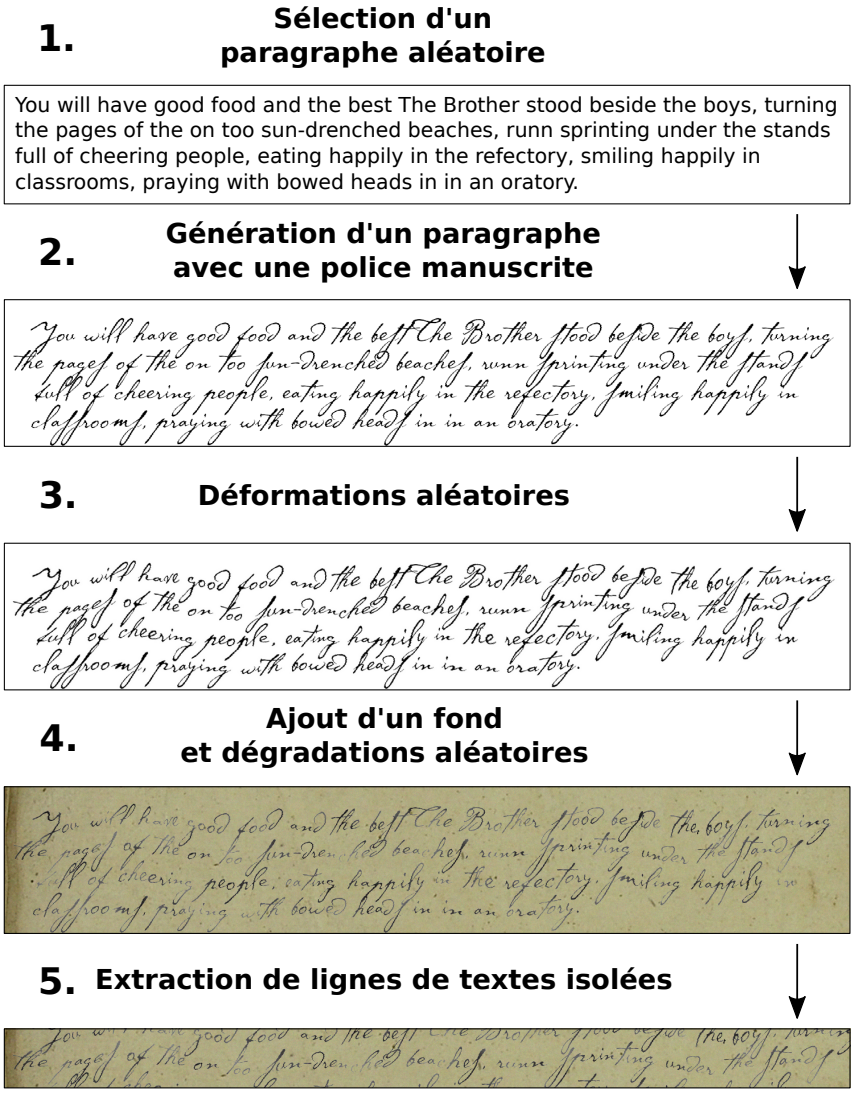


FIGURE 8.4 – Illustration du processus de génération de lignes synthétiques anciennes en cinq étapes.

8.2.2 Contenus textuels anciens

Pour modéliser correctement la langue complexe et variée des documents anciens, il semble important de générer du contenu textuel représentatif de la nature des documents. Pour des documents anciens, il est en revanche rare de trouver un contenu précis avec la même langue, la même époque, ou encore le même auteur. Pour ce faire, nous utilisons le contenu textuel de livres (ou *ebooks*) disponibles en ligne[7]. Ces livres s'étendent sur une période assez large, englobant à la fois de l'écriture similaire aux documents anciens que nous visons à reconnaître, mais aussi des écritures plus récentes. Nous utilisons uniquement

7. Les données utilisées sont disponibles à l'adresse `https://www.gutenberg.org/`.

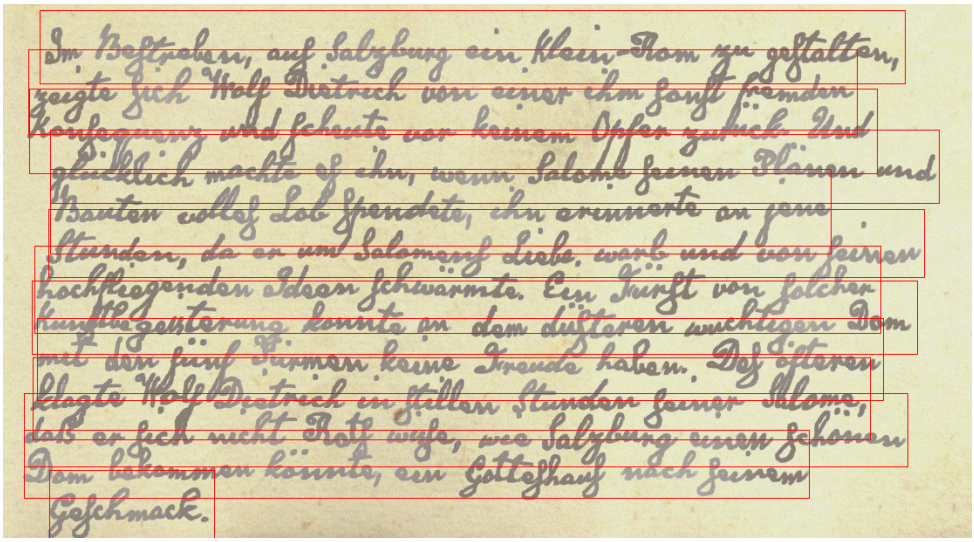


FIGURE 8.5 – Illustration des boites englobantes calculées automatiquement lors de la génération de données synthétiques.

des livres dans les langues des jeux de données anciens étudiés (section 5.1.2, p. 74). Cela nous offre une quantité de contenus textuels assez large, tout en fournissant des textes relativement proches des données réelles étudiées.

Comme pour les données provenant d'articles Wikipédia, nous procédons à un pré-traitement visant à enlever des informations non relatives au contenu textuel. Nous enlevons ainsi : les licences ; les remerciements ; les notes en pied de pages et commentaires éventuels ; ainsi que diverses balises relatives au format. La table 8.2 montre quelques chiffres clés sur le contenu textuel présent dans ces livres, après une étape de pré-traitement.

TABLE 8.2 – Statistiques sur les livres utilisés pour générer des données synthétiques anciennes. Les chiffres sont donnés par langue, et après une phase de pré-traitement pour ne conserver que le contenu textuel exploitable.

	# livres	# paragraphes	# mots
Anglais	2 694	1 720 629	151 152 022
Allemand	780	42 079	3 871 297
Danois	94	51 731	910 013
Suédois	228	42 869	632 134

De plus, nous utilisons, comme pour la génération de données synthétiques modernes, le contenu textuel de la base d'entraînement du jeu de données. Nous utilisons la vérité

terrain à un niveau paragraphe, afin de générer des paragraphes complets (précédant une étape de segmentation en lignes).

Enfin, nous faisons aussi usage du contenu textuel présent dans des articles Wikipédia. Ce contenu textuel est récent et éloigné de celui présent dans des documents anciens réels. Néanmoins, il permet de fournir du contenu textuel varié et en grande quantité pour entraîner notre architecture à modéliser la langue complexe de ces données. Nous pensons que nos architectures de Transformer peuvent ainsi tirer parti de contenu additionnel pour s'entraîner.

8.2.3 Polices d'écritures manuscrites anciennes

Pour générer des écritures manuscrites à partir du contenu textuel, nous utilisons des polices d'écritures au style manuscrit et ancien. Nous avons pour cela parcouru différentes polices d'écritures correspondant à ces critères [8], et nous avons procédé à une sélection manuelle pour en retenir au final 21 proches des écritures des jeux de données que nous visons à reconnaître [9].

Comparées aux polices d'écritures modernes, les polices d'écritures anciennes utilisées offrent bien plus de variabilités grâce à une utilisation plus importante de caractéristiques de polices. Parmi ces caractéristiques, certaines sont en particulier spécifiques à des écritures anciennes, et permettent de générer des écritures proches et réalistes. C'est le cas par exemple de la caractéristique "`hist`", qui permet de générer des variantes stylistiques historiques de caractères, ou la caractéristique "`hlig`" qui permet de recréer des ligatures au style ancien. Ces variations sont illustrées dans la figure 8.6.

8.2.4 Augmentations des images de paragraphe

Enfin, nous appliquons diverses augmentations sur l'image d'écriture générée. La majorité des transformations utilisées sont identiques à celles présentées dans la section 8.1.4 pour des écritures modernes. Cependant, certaines augmentations sont ajoutées où modifiées.

8. Les polices d'écritures sont disponibles en ligne sur `https://fonts.google.com`, `https://www.dafont.com` et `https://www.p22.com`.

9. La liste des 21 polices est disponible avec le code : `https://gitlab.inria.fr/intuidoc-public/synthetic-handwriting-generation`.

(a) Image de texte générée sans caractéristique activée.

(b) Résultat avec la caractéristique "`hist`" activée, qui ajoute des variantes de caractères historiques (pour transformer les "`s`" en "`s` long").

(c) Résultat avec la caractéristique "`hlig`" activée, qui ajoute des ligatures historique. Les différences sont entourées en rouge pour plus de clarté.

FIGURE 8.6 – Exemple d'images de lignes de texte obtenues en activant différentes caractéristiques historiques sur la police d'écriture "P22 Cezanne Pro". Ici, aucune transformation n'est appliquée aux images.

Déformations élastiques Pour déformer les images de paragraphes, nous utilisons deux déformations élastiques.

Une première, similaire à celle proposée dans le cadre des documents modernes vise à déformer l'image localement, et à simuler des variations intra-scripteur. Nous utilisons une grille carrée de contrôle avec comme taille $l = \frac{h}{3}$, où h est la hauteur d'une ligne de texte. De même que sur les documents modernes, nous autorisons des déplacements des points de contrôle d'une distance d'au plus $0,2 \times l$, et avec une distance entre deux points de contrôles adjacents comprise dans l'intervalle $[0,8 \times l; 1,2 \times l]$.

Une seconde déformation élastique, à une échelle plus large, permet notamment de produire des variations dans la courbure des lignes. Nous utilisons pour cela une grille carrée avec comme taille des carrés, $l' = h$. Chaque point de contrôle est déplacé d'une distance de sa position initiale d'au plus $0,15 \times l'$, et deux points de contrôle adjacents doivent être éloignés d'une distance dans l'intervale $[0,8 \times l'; 1,2 \times l']$.

Ces deux transformations sont appliquées avec une probabilité de $1,0$ pour permettre un rendu plus réaliste des paragraphes.

Rotation aléatoire de l'image La rotation aléatoire de l'écriture est en plus de cela modifiée, afin de produire des lignes de texte avec un angle plus prononcé, allant jusqu'à 3°.

8.2.5 Dégradations des images de paragraphe

Nous ajoutons additionnellement différentes dégradations propres aux documents anciens. Pour simuler une encre vieillie, nous utilisons une couleur d'encre aléatoire dans les teintes de marron. Nous ajoutons aléatoirement entre 1 et 5 taches d'encre avec une probabilité de 30%, selon les motifs proposés par Journet et al. [Jou+17]. Enfin, nous effaçons l'encre sur certaines parties de l'image en utilisant un bruit de Perlin. Ce filtre peut effacer aléatoirement jusqu'à 60% de l'encre au maximum sur les parties de l'image les plus effacées.

Nous ajoutons ensuite un arrière plan tiré au hasard parmi différents fonds réels, tels que ceux fourni par Journet et al. [Jou+17], ou ceux des jeux de données READ 2016 [San+16], ou READ de l'ICDAR 2017 [San+17]. Comme pour les images de données synthétiques modernes, différentes transformations aléatoires sont appliquées à l'intégralité de l'image, en modifiant les paramètres de luminosité, de netteté, de contraste, ou en ajoutant du bruit sur l'image.

8.3 Exemples de données synthétiques

Dans cette section, nous présentons des exemples de données synthétiques obtenues avec nos approches de génération. Nous discutons aussi brièvement de l'aspect visuel des résultats obtenus. Pour visualiser plus d'exemples de données synthétiques, nous invitons les lecteurs à consulter l'annexe A (p. 245).

8.3.1 Données synthétiques modernes

Dans la figure 8.7, nous présentons quelques résultats de lignes de texte synthétiques modernes obtenues avec notre approche de génération pour des documents modernes.

Grâce à l'utilisation de polices d'écritures manuscrites assez diverses, différents styles d'écriture très différents sont générés, avec par exemple des écritures comportant de fortes liaisons entre les caractères, ainsi que diverses variantes. L'épaisseur aléatoire du tracé permet aussi d'apporter des variations notables dans les images, et permet de simuler l'utilisation de différents stylos.

D'autre part, il est important de noter l'impact des déformations élastiques qui permettent d'apporter du réalisme à l'écriture synthétique. Cela s'illustre par exemple pour

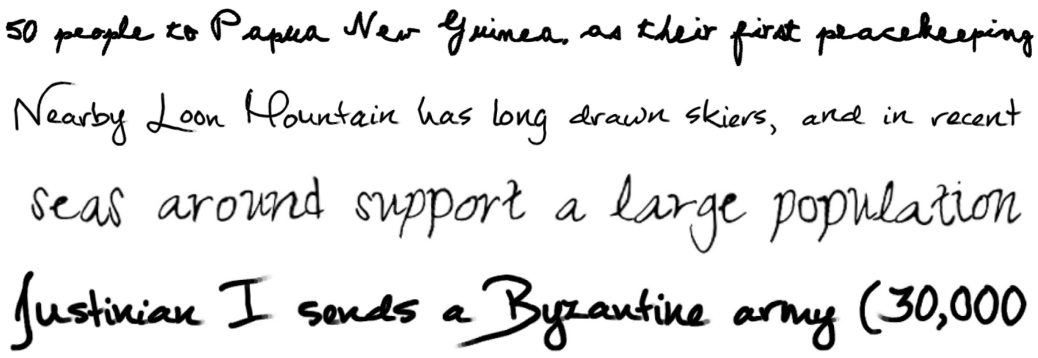


FIGURE 8.7 – Plusieurs exemples d'images de lignes de texte synthétiques manuscrites et modernes obtenues avec notre approche. Ces images visent à imiter le style du jeu de données de reconnaissance de texte manuscrit moderne IAM [MB02], introduit dans la section 2.1.1 (p. 30).

deux caractères identiques générés dans une même ligne de texte, qui auront de petites différences tout en restant similaires.

8.3.2 Données synthétiques anciennes

Nous évaluons maintenant qualitativement les résultats obtenus pour la génération de données synthétiques adaptées aux documents anciens. La figure 8.8 en montre différents exemples.

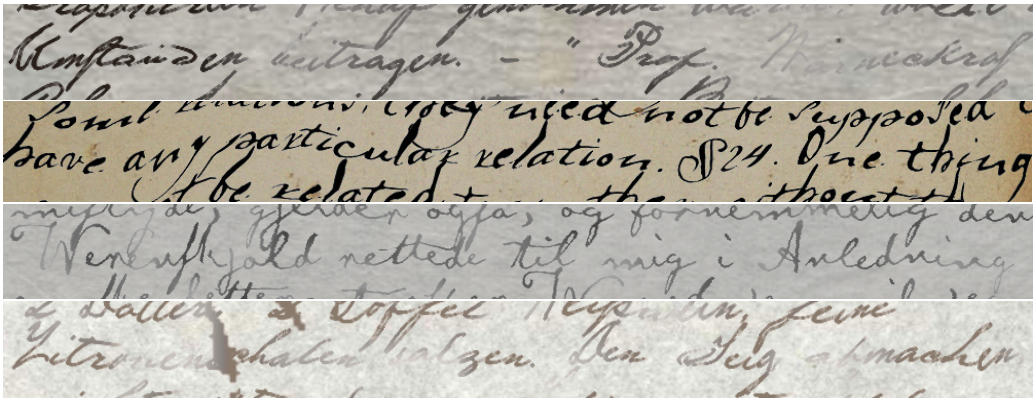


FIGURE 8.8 – Exemples d'images de lignes de texte synthétiques dans un style manuscrit et ancien, obtenues avec notre approche de génération. En particulier, ces images ont été générées dans le but de reproduire le style visuel du jeu de données READ 2018 [Str+18] (introduit dans la section 5.1.2, p. 76).

Au vu des résultats, notre approche semble produire des images de lignes de texte particulièrement réalistes selon nous. Ces résultats sont d'une part rendus possibles par des polices d'écriture manuscrite et au style ancien, ainsi que par les transformations et dégradations utilisées. De même que pour les documents modernes, l'utilisation de

déformations élastiques permet de rendre ces images plausibles en incluant de micro-déformations. L'utilisation de dégradations affectant l'encre, telles que des effacement partiels permet de simuler une encre hétérogène et semblable à celle des documents réels. Enfin, la présence d'enjambements sur les lignes de texte contribue aussi à améliorer la ressemblance avec des écritures réelles.

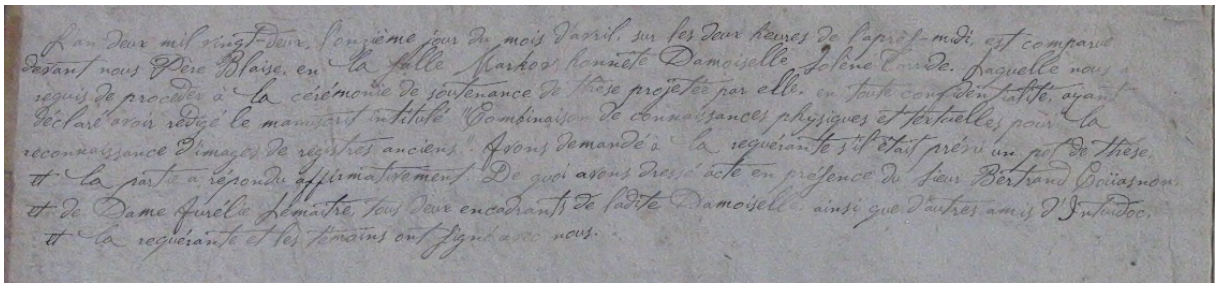


FIGURE 8.9 – Un exemple de paragraphe synthétique dans le style d'un document ancien.

Il est à noter que notre approche permet aussi de générer des paragraphes entiers de données synthétiques dans le style des documents anciens. La figure 8.9 illustre un exemple de paragraphe complet généré. Ces paragraphes semblent de même réalistes et pourraient notamment servir à d'autres contextes applicatifs tels que la reconnaissance de paragraphes ou de pages.

8.4 Discussion

Dans ce chapitre, nous avons présenté deux approches pour générer des données synthétiques manuscrites :

— l'une pour générer des lignes de texte modernes ;

— l'autre pour générer des lignes de texte dans le style des documents anciens, basée sur la génération de paragraphes.

Nous avons ici donné une description détaillée des différents procédés, et des paramètres, dans le but de permettre une reproduction de notre méthode.

Nos approches permettent d'obtenir des images qui nous semblent réalistes. C'est en particulier le cas pour des documents anciens qui sont pourtant bien plus complexes à générer. Notre approche est, à notre connaissance, la première à proposer des données synthétiques à base de polices d'écriture pour les documents anciens. Des visuels additionnels sont en particuliers disponibles dans l'annexe A (p. 245).

Dans le chapitre 10, nous évaluerons l'impact de ces données synthétiques modernes sur l'entraînement de nos architectures. Plus tard, dans le chapitre 11, nous ferons de même dans le cadre plus complexe de la reconnaissance d'écritures manuscrites dans des documents anciens.

Il est important de noter que notre approche ne se limite cependant pas à ces deux scénarios. En effet, elle peut facilement être adaptée à différents scénarios comme par exemple la reconnaissance d'écriture imprimées modernes ou anciennes. Il suffit alors de fournir des polices d'écritures adéquates, ainsi que d'adapter éventuellement les paramètres des différentes augmentations et dégradations. De plus, notre méthode peut aussi servir à générer des paragraphes complets d'écritures manuscrite dans le style des documents modernes, ou anciens, pour aider l'apprentissage d'architectures de reconnaissance d'écriture à des échelles plus larges [YB20 ; CCP21 ; CCP22 ; SK21 ; CCP23]. L'annexe A montre en particulier des visuels de paragraphes modernes qui peuvent être obtenus par notre approche.

L'algorithme de synthèse de données modernes a fait l'objet d'un article présenté au workshop international DAS 2022 [Bar+22b]. La synthèse de données au style des documents anciens est quant à elle présentée dans un article en cours de relecture dans la revue internationale IJDAR [Bar+23b]. De plus, ces algorithmes ont aussi fait l'objet de communications dans des congrès francophones [Bar+21a ; Bar+22a ; Bar+22c ; Bar+23a].

Chapitre 9

Stratégies d'entraînement et de prédiction

Dans ce chapitre, nous proposons deux stratégies dans le but de réduire les taux d'erreurs de nos modèles entraînés avec peu de données annotées :

— une stratégie d'entraînement et de spécialisation sur des documents avec très peu de données annotées, dans la section 9.1 ;

— ainsi qu'une stratégie de prédiction à base d'augmentations en test sur de multiples prédictions et d'une fusion des prédictions, dans la section 9.2.

Les stratégies que nous proposons ne sont pas limitées à des architectures de Transformer, et peuvent bénéficier à de nombreuses autres architectures. Concernant la stratégie d'entraînement, celle-ci se divise en trois parties que nous verrons en détail. Les deux premières parties peuvent bénéficier à l'ensemble des architectures, tandis que la troisième partie est quant à elle applicable uniquement à des architectures encodeur-décodeur. Enfin, la stratégie de prédiction peut être utilisée pour réduire les erreurs de n'importe quelle approche de reconnaissance d'écriture.

9.1 Stratégie d'entraînement et de spécialisation avec peu de données annotées

Contexte

Comme nous l'avons vu dans le chapitre 5, les jeux de données dans le domaine de la reconnaissance d'écriture manuscrite sont en général limités à très peu d'exemples. C'est en particulier le cas des documents anciens qui sont bien plus complexes à annoter. Différentes approches (que nous avons détaillées dans le chapitre 5) sont utilisées pour entraîner efficacement les modèles avec peu de données, comme par exemple l'augmentation ou la synthèse de données.

D'autres approches proposent de changer la procédure d'entraînement pour faire face au manque de données annotées. C'est en particulier le cas du jeu de données READ 2018 [Str+18] que nous avons introduit dans la section 5.1.2 (p. 76). Pour obtenir des taux d'erreurs faibles sur des documents spécifiques contenant très peu d'annotations (entre 20 et 750 lignes annotées), il est proposé de commencer par entraîner un modèle générique avec une quantité importante de données annotées variées (environ 10 000 lignes). Ce modèle est ensuite utilisé comme base pour une adaptation pour des données spécifiques.

De manière similaire, Swaileh et al. [SSP19] proposent de combiner les jeux de données IAM [MB02] et RIMES [Aug+06], écrits en anglais et en français respectivement, pour entraîner un reconnaisseur de texte, ainsi qu'un modèle de langue, tout deux génériques. La même procédure est réalisée par Bluche et al. [BM17], mais en combinant beaucoup plus de données dans des langues différentes. Ils utilisent notamment différents jeux de données existants comme IAM et RIMES, ainsi que des données privées. Ingle et al. [Ing+19] font de même avec leurs propres données. Pour entraîner des architectures de Transformer, des données externes sont fréquemment utilisée en plus de jeux de données existants et des données synthétiques. Singh et al. [SK21] et Li et al. [Li+23] combinent ainsi des données externes publiques pour entraîner leurs modèles, tandis que Diaz et al. [Dia+21] utilisent des données privées pour entraîner leurs architectures.

D'autre part, Michael et al. [Mic+19] proposent d'entraîner leur modèle à la fois sur le jeu d'apprentissage et sur le jeu de validation. Ils parviennent ainsi à réduire le CER de 7 à 11% sur les divers jeux de données étudiés.

Résumé de la stratégie d'entraînement

Dans le but de transcrire des documents spécifiques sur lesquels peu de données sont annotées, nous proposons, en tant que contribution de la thèse, une stratégie d'entraînement inspirée de ces diverses approches. Nous la concevons pour qu'elle bénéficie aux architectures de Transformer, mais elle peut aussi bénéficier à de nombreuses architectures.

La stratégie d'entraînement et de spécialisation que nous proposons a pour but de s'adapter au style propre à un auteur spécifique, tout en conservant un modèle de langue générique. Pour cela, nous jouons sur les différents leviers à notre disposition, c'est-à-dire : des données génériques ; des données spécifiques ; des données synthétiques ; ainsi que l'entraînement des modèles. Elle se divise en 3 étapes principales :

1. un entraînement sur un vaste ensemble de données variées, avec des ensembles d'entraînement et de validation séparés ;
2. un second entraînement sur ce même ensemble, en incluant les données de validation dans les données d'apprentissage ;
3. une spécification sur des données spécifiques, en réentraînement l'encodeur et en gardant les poids du décodeur figé.

La table 9.1 résume ces différentes étapes que nous allons maintenant détailler.

TABLE 9.1 – Résumé des différentes étapes qui composent notre stratégie d'entraînement et de spécialisation. Nous indiquons notamment comment sont utilisés les ensembles d'entraînement et de validation des données génériques (G), ainsi que spécifiques (S). De plus, nous précisons quelles parties des modèles sont entraînées à chaque étape, l'utilisation de données synthétiques, ainsi que le critère d'arrêt utilisé.

Étape	Données d'entraînement	Données de validation	Données synthétiques	Entraînement de l'encodeur	entraînement du décodeur	Critère d'arrêt
1	entraînement G	validation G	✓	✓	✓	arrêt précoce
2	entraînement G + validation. G	-	✓	✓	✓	quelques époques
3	entraînement S	validation S	-	✓	figé	arrêt précoce

9.1.1 Entraînement sur un jeu de données générique

Comme nous visons à entraîner un modèle sur des données spécifiques contenant très peu de données annotées, en tant que première étape, nous commençons par pré-entraîner nos architectures sur un jeu de données générique et contenant un nombre bien plus important d'annotations. Même si les documents diffèrent des données spécifiques (sur la langue utilisée ou sur la période par exemple), nous nous attendons tout de même à une amélioration de nos modèles en effectuant un tel pré-entraînement [BM17 ; Str+18].

De plus, nous proposons l'utilisation de données synthétiques variées pour compléter le jeu de données générique. Cela permet ainsi d'ajouter des données annotées obtenues à moindre coût en comparaison à des données réelles. Ces données synthétiques sont générées à la volée durant l'entraînement, comme expliqué dans le chapitre 8, ce qui permet d'obtenir des données différentes à chaque itération.

Pour entraîner nos architectures sur ces données génériques, 10% des données d'apprentissages sont réservées comme ensemble de validation. Enfin, nous utilisons de l'arrêt précoce pour mettre fin au processus d'apprentissage lorsque nous observons un certain nombre d'époques (typiquement 50–200) sans amélioration sur l'ensemble de validation.

9.1.2 Entraînement en utilisant toutes les données génériques

Dans la seconde étape de la stratégie d'entraînement, nous visons à maximiser les données d'entraînement vues par le modèle. Nous proposons ainsi de continuer l'entraînement du meilleur modèle obtenu lors de l'étape 1, en incluant l'ensemble de validation dans les données d'entraînement existantes. De plus, nous continuons à utiliser de la génération de données synthétiques durant cette seconde étape.

Cependant, pour mitiger un effet de surapprentissage, nous entraînons le modèle sans validation pendant un nombre limité d'époques (typiquement une dizaine d'époques).

Ces deux premières étapes ont pour but l'entraînement d'un modèle générique, adapté à la fois pour reconnaître différents styles visuels et pour modéliser différents types de langues. Un tel modèle est alors probablement pertinent comme point de départ pour une spécialisation sur des données spécifiques.

9.1.3 Spécialisation au style visuel des données spécifiques

Enfin, dans l'étape 3 de notre stratégie d'entraînement, nous visons à spécialiser le modèle sur des données spécifiques avec très peu de données annotées. De la même manière que pour l'étape 1, nous réservons 90% des données annotées disponibles pour l'entraînement du modèle, et 10% pour la validation. Nous utilisons de plus de la validation croisée pour les ensembles avec très peu de données annotées, ce qui permet notamment de limiter l'impact d'un ensemble de validation peu fiable.

Pour les architectures encodeur-décodeur (comme c'est le cas pour les architectures VLT et LT), nous adaptons uniquement la partie encodeur. Cela permet en particulier d'entraîner le modèle à se spécialiser sur le style visuel d'un auteur et des documents spécifiques. Les poids du décodeur restent quant à eux figés, ce qui permet de conserver des capacités de modélisation de la langue générique. Cela se révèle utile, car nous avons observé que le décodeur est particulièrement sujet à du surapprentissage. Il n'est donc pas intéressant de spécialiser le décodeur sur le peu de données annotées disponibles. Nous aborderons ce point plus tard, dans le chapitre 11.

Durant cette étape, nous n'utilisons pas de données synthétiques dans le but de spécialiser le modèle sur le style visuel précis des données spécifiques. De même que pour l'étape 1, nous utilisons de l'arrêt précoce, et nous conservons le modèle ayant obtenu le CER le plus faible sur l'ensemble de validation.

9.2 Stratégie de prédictions à base d'augmentation en test et de vote

Soullard et al. [Sou+19] ont montré que les réseaux de neurones sont sensibles à des variations dans des images de ligne de texte. Pour résoudre ce problème et réduire les taux d'erreurs, une solution consiste à consolider les prédictions en utilisant des augmentations de données en test, combinée à un mécanisme de vote parmi les multiples prédictions [Str+18 ; Dut+18 ; Sou+19]. Nous en avons en particulier discuté dans la section 5.3.2 (p. 85).

Soullard et al [Sou+19] proposent de générer de multiples images à partir d'une unique image réelle. Celles-ci sont augmentées aléatoirement avant d'être passées en entrée de leur modèle pour obtenir de multiples prédictions. Ces prédictions sont ensuite alignées au niveau caractère en utilisant un algorithme inspiré de l'approche ROVER [Fis97]. Une prédiction finale est ensuite obtenue au moyen d'un processus de vote.

Stratégie de prédiction proposée

Nous proposons, en tant que contribution de cette thèse, une version améliorée de l'algorithme de prédiction proposée par Soullard et al [Sou+19], dans le but d'obtenir une prédiction plus robuste et contenant probablement moins d'erreurs. Les principales différences se situent dans la phase d'alignement. D'une part, les prédictions sont filtrées pour évincer des prédictions trop éloignées. D'autre part, les prédictions sont insérées et alignées par ordre croissant de difficulté.

Grâce à l'utilisation d'un filtre pour supprimer les prédictions trop éloignées, notre méthode aura alors tendance à ne pas être impactée par des prédictions aberrantes. Elle a ainsi l'avantage de pouvoir être utilisée sur des modèles sujets à du surapprentissage, là où d'autres méthodes rencontreraient des difficultés lors de l'alignement. De plus, notre stratégie passe mieux à l'échelle, et peut être utilisée avec des grandes quantités de prédictions pour une même image réelle.

Il est à noter que notre stratégie peut s'appliquer indépendamment du modèle de reconnaissance utilisé.

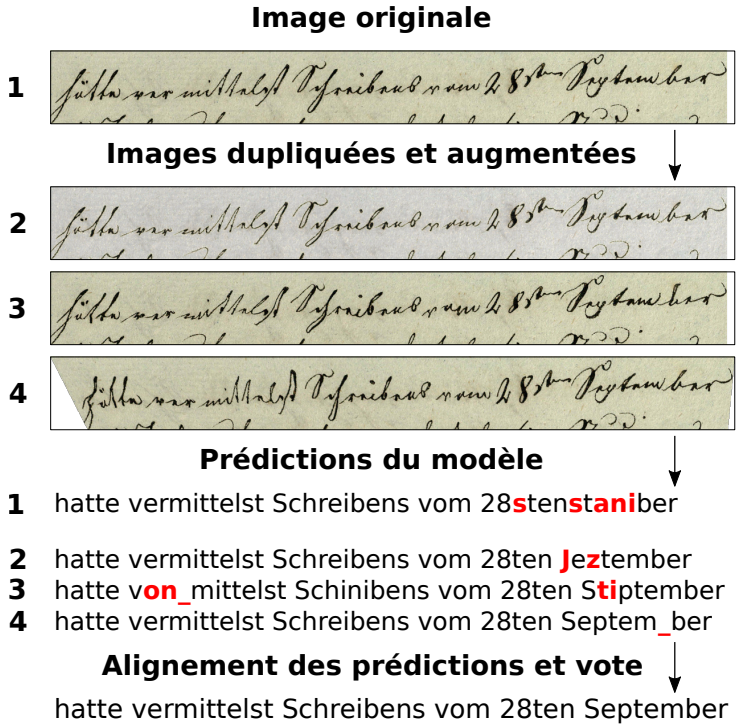


FIGURE 9.1 – Illustration du processus complet de la stratégie de prédiction proposée. Des images augmentées aléatoirement sont obtenues à partir d'une unique image. Les prédictions associées, contenant différentes erreurs, sont fusionnées au moyen d'une étape d'alignement ainsi qu'une étape de vote pour obtenir une prédiction finale correcte.

La figure 9.1 illustre le processus complet réalisé par notre stratégie de prédiction. Chaque image réelle est augmentée aléatoirement $n-1$ fois, en utilisant les mêmes augmentations que celles utilisées lors de l'entraînement (*c.f.* la section 10.1.2, p. 152). Il en résulte donc un total de n images, en incluant l'image réelle originale. Les images sont ensuite passées en entrée d'un modèle de reconnaissance d'écriture pour obtenir un ensemble de n prédictions $P = \{p_i \mid i \in [\![1; n]\!]\}$. Il est en revanche à noter que l'obtention de n prédictions au lieu d'une seule prédiction demande un temps plus important et croissant avec n, ce qui ralentit en conséquence le processus de prédiction. Pour fusionner les prédictions, nous utilisons ensuite l'algorithme 1, basé sur un alignement et un vote des prédictions.

Dans un premier temps, la fonction DISTANCEEDITIONMOYENNE calcule pour chaque prédiction $p_i \in P$, la distance d'édition moyenne d_i entre la prédiction p_i et toute autre

Algorithme 1 Algorithme d'alignement et de vote des prédictions

Entrée : P, l'ensemble des prédictions et τ, un seuil de rejet
Sortie : v, la prédiction finale votée

$D \leftarrow \textsc{DistanceEditionMoyenne}(P)$
$P, D \leftarrow \textsc{FiltreSeuil}(P, D, \tau)$
Trier P selon les valeurs de D
$A \leftarrow \emptyset$ ▷ A est l'ensemble des prédictions alignées
pour $p_i \in P$ **faire**
    $A \leftarrow \textsc{InsertionEtAlignement}(p_i, A)$
fin pour
$v \leftarrow \textsc{Vote}(A)$
renvoyer v

prédiction dans P. On a alors l'ensemble D défini tel que :

$$D = \{d_i \mid i \in [\![1; n]\!]; d_i = \textsc{DistanceEditionMoyenne}(p_i, P \setminus \{p_i\})\}$$
$$\textsc{DistanceEditionMoyenne}(p_i, P') = \frac{\sum_{p_j \in P'} \textsc{DistanceEdition}(p_i, p_j)}{|P'|}$$

Nous utilisons ensuite un seuil τ pour filtrer l'ensemble des prédictions P sur la base des valeurs des distances d'édition moyennes dans D. Si une prédiction p_i a une distance d'édition moyenne d_i telle que $d_i > \tau$, alors nous supprimons cette prédiction de l'ensemble P, ainsi que la distance associée d_i dans D. Intuitivement, si une prédiction obtient une distance d'édition moyenne élevée, cette prédiction diffère alors considérablement des autres prédictions. La supprimer permet alors la production d'une prédiction finale plus pertinente et robuste à des prédictions aberrantes. Dans nos expériences, nous utilisons en particulier la valeur $\tau = 0{,}75$. Ici, τ autorise donc jusqu'à 75% ce caractères erronés.

L'ensemble des prédictions retenues sont mises à la même longueur, en insérant dans l'ensemble A, et en alignant successivement les prédictions $p_i \in P$. Pour cela, nous utilisons la fonction $\textsc{InsertionEtAlignement}(p_i, A)$. En particulier, l'ensemble des prédictions P est tout d'abord trié par ordre croissant des distances d'édition moyennes dans D. Les prédictions sont donc prises en compte dans l'ordre croissant de difficulté, en commençant par les prédictions les plus proches en moyenne des autres.

L'alignement des des prédiction $p_i \in P$ avec l'ensemble des prédictions alignées A, est réalisé au moyen de l'algorithme d'alignement proposé par Soullard et al. [Sou+19]. Il se base en particulier sur l'algorithme du calcul de la distance d'édition, et permet de trouver un nombre minimal de changements au niveau des caractères, en utilisant un jeton

joker '∗' pour indiquer l'insertion d'un caractère. L'étape d'alignement est illustrée dans la figure 9.2.

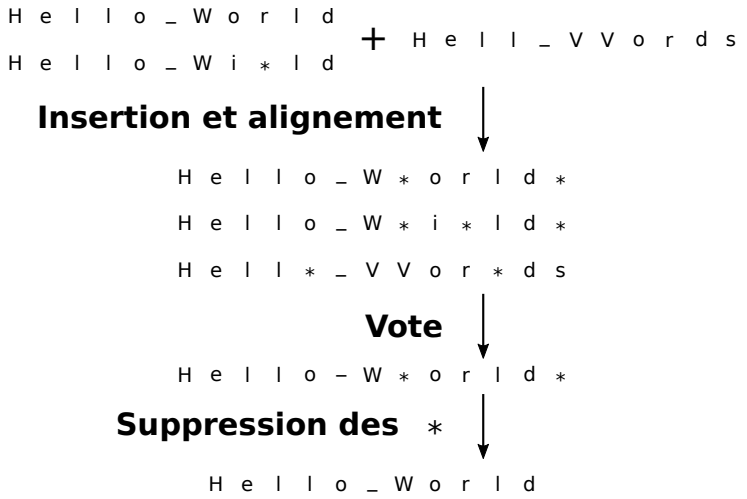


FIGURE 9.2 – Exemple d'alignement et de vote obtenu en insérant la séquence "`Hell VVords`" dans les séquences alignées "`Hello World`" et "`Hello Wi∗ld`".

Enfin, une fois toutes les prédictions alignées, et donc de tailles identiques, un vote est effectué (fonction VOTE de l'algorithme 1) afin de déterminer le caractère majoritaire à chaque position des séquences alignées. Une fois les jetons joker '∗' enlevés, la prédiction votée v est alors obtenue. Cette procédure est illustrée dans la figure 9.2.

9.3 Discussions

Dans ce chapitre, nous avons présenté deux contributions de la thèse :

— une stratégie d'entraînement sur des données génériques, et de spécialisation de certaines couches au style spécifique d'un document, tout en conservant un modèle de langue générique ;

— ainsi qu'une stratégie de prédiction basée sur des augmentations en test, un algorithme d'alignement des prédictions au niveau des caractères, ainsi qu'un système de vote pour consolider les prédictions produites par un modèle, et réduire le nombre d'erreurs.

La stratégie d'entraînement est principalement dédiée à des modèles encodeur-décodeur, mais certaines parties restent cependant applicables à n'importe quel système de reconnaissance d'écriture. La stratégie de prédiction peut, pour sa part, être utilisée par n'importe quel système.

Dans les sections suivantes, nous évaluerons l'impact de nos différentes stratégies. Ainsi, dans le chapitre 11, nous montrerons que la stratégie d'entraînement est utile pour se spécialiser sur des documents spécifiques avec peu de données annotées, comme c'est le cas du jeu de données READ 2018. Dans les chapitres 10 et 11, nous verrons que la stratégie de prédiction permet une réduction efficace des taux d'erreur.

Ces résultats font l'objet d'un article de journal en cours de relecture dans la revue internationale IJDAR [Bar+23b].

Troisième partie

Résultats expérimentaux

Dans cette partie, nous évaluons les différentes contributions de la thèse, à savoir :

— les différentes architectures introduites, c'est-à-dire, les architectures CFTNN (chapitre 6), VLT et LT (chapitre 7) ;

— l'algorithme de génération de données synthétiques (introduit dans le chapitre 8), dans le cadre de documents modernes, ainsi que pour des documents anciens ;

— les stratégies d'entraînement et de prédiction proposées dans le but de réduire les taux d'erreur (chapitre 9).

Pour cela, nous utiliserons les jeux de données modernes IAM et RIMES (introduits dans la section 2.1, p. 29), ainsi que les jeux de données anciens READ 2016 et READ 2018 (présentés dans la section 5.1.2, p. 74). Nous proposerons par ailleurs des comparaisons avec les différentes approches de l'état de l'art.

Dans le chapitre 10, nous commencerons par valider l'intérêt des différentes contributions de la thèse sur des documents modernes. Le chapitre 11 présentera les résultats que nous obtenons avec nos approches sur des documents anciens plus complexes et dont le nombre d'exemples annotés pour l'apprentissage est restreint.

Chapitre 10

Validation du modèle sur des documents modernes

Dans ce chapitre, nous évaluons les architectures présentées dans les chapitres précédents sur des lignes de texte issues de documents modernes. Comparés à des documents anciens, les documents modernes sont plus simples à reconnaître et possèdent plus de données annotées. Nous utilisons ainsi les documents modernes pour valider nos architectures, tout en démontrant qu'elles sont capables d'obtenir d'excellents résultats sur ceux-ci. De plus, nous présentons dans ce chapitre diverses conclusions tirées des résultats présentés.

Nous proposons différentes expériences réalisées sur le jeu de données d'écriture manuscrite anglaise modernes IAM [MB02]. En fin de chapitre, nous comparons aussi les résultats obtenus par nos approches avec l'état de l'art sur le jeu de données d'écritures manuscrites françaises : RIMES [Aug+06].

Les différents résultats de ce chapitre seront présentés en comparant des modèles entraînés sans données synthétiques, ainsi qu'avec données synthétiques. Les données synthétiques étant une contribution importante de la thèse en plus des architectures proposées, nous procédons de la sorte afin de fournir une analyse poussée de l'impact de l'utilisation de données synthétiques sur différents scénarios. Additionnellement, cela permet de se comparer de manière équitable avec la majorité des approches de l'état de l'art.

Nous tenons à mettre un accent particulier sur la reproductibilité de nos résultats. En plus de notre code accessible en ligne [1], nous détaillerons ainsi les différents paramètres de nos entraînements. De plus, nous rendons disponible l'intégralité des poids de nos modèles [1].

Nous présentons le protocole expérimental dans la section 10.1. Dans la section 10.2, nous démontrons les capacités de notre architecture convolutive à Transformer rapide. Nous étudions ensuite les architectures de Transformer autorégressives et leurs excellents

1. Notre code, ainsi que les poids des modèles sont disponibles à l'adresse suivante : `https://gitlab.inria.fr/intuidoc-public/vlt`

résultats dans la section 10.3. La section 10.4 présente quant à elle l'intérêt des stratégies proposées pour entraîner nos architectures. Nous validons nos approches dans des scénarios avec peu de données annotées dans la section 10.5. La section 10.6 présentera des résultats obtenus avec notre stratégie d'évaluation, et dans la section 10.7, nous comparons nos approches avec l'état de l'art sur les jeux de données IAM et RIMES. Nous terminerons ce chapitre par une brève analyse des prédictions de nos architectures dans la section 10.8, suivie d'une synthèse des résultats dans la section 10.9.

10.1 Protocole expérimental

Dans cette section, nous présentons notre protocole expérimental dans le but de rendre les expériences reproductibles.

10.1.1 Jeux de données

Dans ce chapitre, nos différentes expériences sont réalisées sur le jeu de données d'écritures manuscrites anglaises modernes IAM [MB02]. Ce jeu de données a été introduit dans la section 2.1.1 (p. 30). Nous l'utilisons comme il est fortement considéré par la communauté pour se comparer. Nous utilisons en particulier la partition "Aachen".

De plus, nous nous comparons aussi avec l'état de l'art sur le jeu de données RIMES [Aug+06] qui a pour but d'évaluer la qualité de reconnaissance de texte sur des images d'écriture manuscrite française et moderne. Il a été présenté dans la section 2.1.2 (p. 31).

10.1.2 Traitement des données

Pour augmenter la variabilité de nos modèles, nous utilisons de l'augmentation de données ainsi que de la génération de données synthétiques.

Augmentations de données

Lors du processus d'apprentissage, chaque donnée réelle est augmentée aléatoirement. Pour ce faire, nous appliquons différentes transformations aléatoirement. Chaque augmentation est ainsi appliquée avec une probabilité de 0,2. Nous décrivons maintenant les augmentations de données utilisées.

Dilatation et érosion de l'image L'image de texte subit aléatoirement une dilatation ou de l'érosion similaire au processus décrit dans la section 8.1.4 (p. 124). Un filtre est ainsi utilisé pour affiner ou grossir le tracé du texte. Contrairement au processus de génération de données synthétiques, nous utilisons uniquement les deux premiers filtres qui auront un effet plus faible. De plus, aucune étape préalable de binarisation n'est réalisée. La couleur du fond est ainsi affectée légèrement sans dégrader les performances de l'architecture. Au contraire, cela provoque un effet similaire à un changement de luminosité de l'image, qui peut apporter plus de variations.

Distorsions élastiques Nous appliquons ensuite une distorsion élastique sur l'image de ligne de texte. De même que la transformation appliquée lors de la génération de données synthétiques réelles (présentée dans la section 8.1.4, p. 124), nous divisons l'image en un pavage régulier, avec chaque carré de longueur $l = \frac{H}{5}$, où H est la hauteur de l'image. Chaque point de contrôle de la grille est alors déplacé d'une distance aléatoire d'au plus $0,2 \times l$ dans chaque direction. Aucune contrainte entre des points de contrôles adjacents n'est appliquée.

Transformation de la perspective Nous changeons la perspective de l'image en déplaçant aléatoirement les coins de l'image de manière identique à ce qui est présenté dans la section 8.1.4 (p. 124). Cela permet ainsi de simuler divers points de vue sur l'image, mais aussi des rotations ou inclinaisons de l'image. Lorsque cette transformation est appliquée, nous comblons les zones vides de l'image augmentée avec un fond blanc uniforme.

Déplacement horizontal du texte Nous rajoutons des colonnes de pixels sur les bordures de l'image. Pour ce faire, sur les côtés gauche et droit de l'image un fond blanc uniforme d'une longueur aléatoire allant jusqu'à $\frac{H}{3}$ est appliqué. Cela permet ainsi de simuler le placement du texte à différents endroits de l'image.

Application d'un bruit Gaussien Enfin, nous appliquons un bruit Gaussien sur les pixels de l'image. Pour une image centrée-réduite, le bruit Gaussien ajoute aux valeurs des pixels une valeur aléatoire tirée selon la loi normale de paramètre $\mathcal{N}(0; 0{,}02)$.

Génération de données synthétiques

Lors du processus d'apprentissage, nous générons additionnellement des données synthétiques pour fournir une plus grande variabilité ainsi qu'un nouveau vocabulaire. Pour

ce faire, nous générons à chaque époque une quantité de données synthétiques égale au nombre de données réelles. La manière de générer ces données est décrite dans la section 8.1 (p. 120).

Normalisation des données

Enfin, nous appliquons diverses normalisations des données. Ces normalisations sont appliquées que ce soit pour des données réelles et augmentées, ou des données synthétiques. De plus, elles sont à la fois utilisées lors de la phase d'entraînement et lors de la phase de prédiction. Elles sont utilisées lors de la phase d'apprentissage, mais notons qu'elles sont aussi utilisées lors de la phase d'inférence lorsque nous utilisons la stratégie de prédiction proposée (dans la section 10.6 et les suivantes).

Pour des données modernes, où le texte est généralement noir sur fond blanc, nous utilisons alors des images en niveau de gris. L'image est redimensionnée à une hauteur fixe de 128 pixels tout en conservant les proportions de l'image.

Pour un entraînement rapide sur des cartes graphiques, nous utilisons des *batchs* d'images où les images de lignes de texte sont toute de même dimension. De fait, nous ajoutons un *padding* blanc uniforme sur la droite des images pour obtenir des images de même longueur. Tandis que pour des architectures de Transformer le *padding* peut être masqué pour s'assurer que les architectures n'utilisent pas ces valeurs, ce n'est pas le cas de couches récurrentes combinées à des couches convolutives. Ces dernières obtiennent alors des résultats conditionnés par la présence de *padding* dû à différentes tailles de *batch*. Pour annuler cet effet, nous ajoutons alors un *padding* fixe blanc uniforme de 64 pixels sur la gauche et la droite de l'image. Cette valeur a été choisie pour être proche et légèrement supérieure à la largeur du champ réceptif utilisé (46).

Dernièrement, nous normalisons l'ensemble des données, que ce soit en entraînement ou en prédiction. Pour ce faire, nous calculons la moyenne et l'écart-type des valeurs de l'ensemble des pixels dans le jeu d'entraînement. Les données sont ensuite ramenées à des valeurs centrées et réduites selon ces paramètres.

10.1.3 Architectures

Nous présentons ici les différentes architectures qui sont utilisées dans nos expériences.

CRNN (*Convolutional Recurrent Neural Network*) : architecture de réseau de neurones convolutif à récurrence Nous proposons une architecture classique de

CRNN en tant que modèle de base pour se comparer. L'architecture CRNN proposée est fortement inspirée des travaux de Puigcerver [Pui17]. Elle est composée de la base convolutive utilisée dans nos architectures (section 6.1.2, p. 100), ainsi que de 4 couches de LSTM bidirectionnels avec 128 neurones pour chaque direction. Elle possède au total 1,7M paramètres.

CFTNN (*Convolutional Fast Transformer Neural Network*) : architecture convolutive à Transformer rapide Cette architecture est une des contributions de la thèse et a été détaillée dans le chapitre 6. Elle est composée d'une base convolutive ainsi que de 4 blocs d'encodeur Transformer, et reste légère (3,5M paramètres).

VLT (*Very Lightweight Transformer*) : architecture de Transformer très légère L'architecture VLT est une autre contribution de la thèse introduite dans le chapitre 7. Elle dérive de l'architecture MLT qui y est présentée. L'architecture VLT est autorégressive et est composée d'une base convolutive, de 4 blocs d'encodeur Transformer et de seulement 2 blocs de décodeur Transformer. Elle inclut donc des capacités de modélisation de la langue tout en restant très légère (5,6M paramètres).

LT (*Lightweight Transformer*) : architecture de Transformer légère L'architecture LT est une autre variante de l'architecture MLT proposée dans le chapitre 7 en tant que contribution de la thèse. Elle utilise 4 blocs de décodeur Transformer, et reste légère (7,7M de paramètres) en comparaison à d'autres architectures Transformer dans le domaine.

MT (*Medium Transformer*) : architecture de Transformer de taille modérée Nous proposons de comparer nos architectures de Transformer avec une architecture de Transformer plus large dans le but d'évaluer l'impact de nos architectures légères. L'architecture MT est similaire à l'architecture LT, avec 512 neurones au lieu de 256 et 8 têtes d'attention au lieu de 4 dans chaque bloc Transformer. Elle comptabilise un total de 29,9M paramètres, et se place entre les architectures proposées et les architectures larges de Transformer [Kan+22 ; Li+23] pour ce qui concerne le nombre de poids.

Seq2seq : Architecture encodeur-décodeur séquence à séquence récurrente Nous comparons aussi nos architectures de Transformer avec une architecture encodeur-décodeur récurrente, inspirée des travaux de Michael et al. [Mic+19]. Elle utilise l'architec-

ture CRNN en tant qu'encodeur, un décodeur composé d'une couche de LSTM avec 256 neurones, ainsi que de l'attention basée sur celle proposée par Bahdanau et al. [BCB14]. Elle comptabilise un total de 2,3M paramètres.

10.1.4 Procédures d'entraînement et de prédiction

Dans cette section, nous fournissons des détails quant à l'entraînement de nos architectures, ainsi que la manière dont nous évaluons nos architectures. Il est à noter que ces détails sont communs à l'ensemble de nos architectures. Les détails propres à l'entraînement de chaque architecture (comme par exemple l'utilisation d'une fonction de coût hybride) sont quant à eux présentés dans les chapitres respectifs.

Détails de l'entraînement

Nous entraînons nos architectures avec l'optimiseur *Adam* [KB14]. Le taux d'apprentissage est changé dynamiquement lors de l'entraînement, en suivant la politique d'apprentissage proposée par Vaswani et al. [Vas+17]. Durant les 7 premières époques de l'apprentissage (environ 50 000 images de lignes de texte), le taux d'apprentissage est augmenté linéairement jusqu'à une valeur maximale de $0,001$. Le taux d'apprentissage est ensuite réduit exponentiellement durant les époques consécutives selon une décroissance en $\frac{1}{\sqrt{x}}$. La figure 10.1 montre l'évolution du taux d'apprentissage au cours des époques. Nous avons observé que cette planification du taux d'apprentissage permet de régulariser efficacement l'entraînement de nos architectures. Comme proposé par Vaswani et al [Vas+17], la valeur du taux d'apprentissage est cependant multipliée par un facteur en $\frac{1}{\sqrt{D}}$, avec D, la dimension du modèle, c'est-à-dire le nombre de neurones dans les couches de Transformer. La valeur maximale atteinte par l'architecture MT ($D = 512$) proposée est alors de 0,000 7 au lieu de 0,001.

Chaque entraînement d'une de nos architectures est réalisé sur un ensemble de 4 cartes graphiques Tesla V100 de la marque NVIDIA. À chaque époque, l'intégralité des données réelles du jeu de données d'entraînement est vue dans un ordre aléatoire. Dans le cas d'un entraînement avec des données synthétiques, nous générons à la volée autant de données synthétiques que de données réelles dans la base d'entraînement. À la fin de chaque époque, le modèle est évalué sur le jeu de validation, et l'entraînement est arrêté après 200 époques sans avoir observé d'amélioration du CER sur le jeu de validation.

L'entraînement est réalisé sans l'utilisation de la stratégie de réentraînement et d'adaptation à des données spécifiques présentée dans la section 9.1 (p. 139). En effet, cette stra-

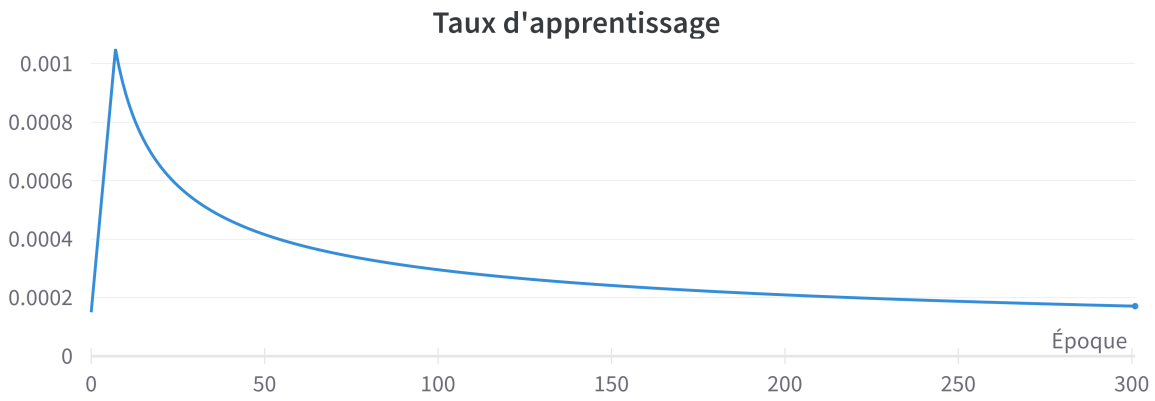


FIGURE 10.1 – Évolution du taux d'apprentissage lors d'un entraînement au fur et à mesure des époques.

tégie est utile pour des spécialisations sur des documents spécifiques avec peu de données annotées.

Procédure d'évaluation

Pour évaluer les résultats de notre architecture, nous utilisons le modèle ayant obtenu le CER le plus faible sur le jeu de validation lors des différentes époques d'entraînement.

Ce modèle est évalué sur le jeu de données de test. Les taux d'erreur finaux (CER et WER comme présentés dans la section 1.4, p. 23), sont obtenus par le biais d'une moyenne des taux d'erreur, pondérée par la longueur de chaque ligne :

$$CER = \frac{\sum_{i=1}^{|\mathcal{D}|} |y_i| \cdot CER_i}{\sum_{i=1}^{|\mathcal{D}|} |y_i|} = \frac{\sum_{i=1}^{|\mathcal{D}|} ED_i}{\sum_{i=1}^{|\mathcal{D}|} |y_i|} \tag{10.1}$$

avec $\mathcal{D}$ le jeu de données de test, $|y_i|$ la longueur (nombre de caractères) de la i-ème vérité terrain, et CER_i le taux d'erreur caractère obtenu entre la prédiction réalisée sur la ligne de texte associée et y_i. De la même manière, ce résultat peut être obtenu en calculant la distance d'édition totale (somme des distances d'édition ED_i) et en la pondérant par la somme des tailles. Cette méthode d'évaluation permet ainsi de représenter fidèlement le nombre d'erreurs commises par les architectures.

Enfin, dans les résultats présentés avec la stratégie de prédiction (section 10.6 et les suivantes), nous procédons à des augmentations en test avec vote, comme expliqué dans la section 9.2 (p. 143). Pour ce faire, chaque image est dupliquée 60 fois, et outre la

première des 60 qui reste inchangée, les 59 autres sont augmentées aléatoirement de la même manière que lors de l'entraînement. Dans nos différentes expériences, nous fixons le seuil de rejet $\tau = 0,75$.

10.2 Validation du modèle convolutif à Transformer rapide

Dans cette section, nous nous concentrons sur le modèle de réseau de neurones convolutif à Transformer rapide (CFTNN) que nous avons proposé dans le chapitre 6. Cette architecture a été proposée dans le but de remplacer les couches de récurrences existantes, pour à la fois produire une meilleure modélisation du contexte ainsi que pour fournir une vitesse de prédiction plus rapide. Pour évaluer ces impacts, l'architecture CFTNN est comparée à l'architecture CRNN.

Dans la section 10.2.1 nous comparons les performances de ces deux architectures. Nous démontrons ensuite dans la section 10.2.2 que l'architecture CFTNN se révèle être particulièrement utile combinée à un décodeur Transformer.

10.2.1 Comparaison des performances de reconnaissance

Taux de reconnaissance

Nous comparons dans un premier temps les taux d'erreur obtenus par les architectures CRNN et CFTNN, entraînées seules et sans pré-entraînement. La table 10.1 montre les taux d'erreur obtenus lors d'un entraînement avec ou sans données synthétiques.

TABLE 10.1 – Comparaison des architectures CRNN et CFTNN entraînées avec ou sans données synthétiques sur le jeu de données IAM.

Architecture	IAM		IAM + données synthétiques	
	CER (%)	WER (%)	CER (%)	WER (%)
CRNN	**6,35**	**22,93**	5,78	**20,90**
CFTNN	6,41	23,88	**5,70**	21,72

Nous observons que les architectures CRNN et CFTNN obtiennent des CER relativement identiques. Ce résultat se retrouve en particulier avec ou sans l'ajout de données synthétiques, bien que l'utilisation de données synthétiques semble bénéficier un peu plus

à l'architecture CFTNN que l'architecture CRNN. Cela peut s'expliquer par le fait que l'architecture CFTNN possède plus de poids.

Nous observons, en revanche, un meilleur WER en utilisant l'architecture CRNN. Ce résultat semble indiquer que l'architecture récurrente est légèrement plus adaptée que l'architecture CFTNN pour retenir un vocabulaire. Bien que les architectures de Transformer permettent une généralisation des couches récurrentes, l'apprentissage des couches Transformer reste complexe. Avec un nombre restreint de poids, l'architecture CRNN semble alors être plus adaptée pour modéliser la langue que l'architecture CFTNN.

Néanmoins, les écarts entre les deux architectures restent relativement proches, que ce soit au niveau du CER, ou du WER. Cela démontre ainsi que notre architecture CFTNN est capable d'obtenir de bons résultats avec un CER proche de 5%, et à un niveau similaire à l'architecture CRNN.

Vitesses d'apprentissage et de prédictions

Nous comparons maintenant la vitesse d'apprentissage et de prédiction des architectures CRNN et CFTNN. Pour ce faire, nous utilisons les versions optimisées implémentées dans la librairie de Pytorch [Pas+19] des couches Transformer et récurrentes. Lors de la réalisation de nos autres expériences, nous utilisons notre propre implémentation de l'architecture Transformer. Celle-ci nous offre en particulier une flexibilité plus importante dans les expérimentations. Il est cependant à noter que nous obtenons des résultats similaires (vitesses et performances) en utilisant notre propre implémentation ou celle proposée dans la librairie Pytorch.

Nous rappelons que l'entraînement de nos architectures est arrêté après 200 époques sans améliorations sur l'ensemble de validation. Pour évaluer le temps de prédiction des architectures, nous évaluons les performances sur l'ensemble du jeu de test d'IAM. Les prédictions sont réalisées par *batch* de 64 images, et nous fournissons les moyennes et écart-types des temps nécessaires pour évaluer un *batch* d'image. Il est à noter que l'évaluation d'un *batch* d'image ou d'images isolées est sensiblement le même. Nous omettons volontairement le premier *batch* qui inclut un temps supplémentaire dû au chargement du modèle. De plus, nous évaluons uniquement le temps mis par les modèles pour fournir des prédictions. La procédure de décodage n'est pas prise en compte, car elle est commune aux deux architectures. La table 10.2 présente les résultats liés au temps d'entraînements et de prédiction des deux architectures.

TABLE 10.2 – Comparaison des temps d'apprentissage et de prédiction entre l'architecture CRNN et l'architecture CFTNN. Les temps d'apprentissage correspondent à une moyenne calculée sur plusieurs expériences. Les temps de prédiction ont quant à eux été mesurés sur plusieurs *batchs*.

Architecture	# paramètres	Temps d'entraînement (en h)	Temps pour évaluer un *batch* (en s)
CRNN	1,7M	**19**	**0,025 ± 0,005**
CFTNN	3,5M	22	0,029 ± 0,004

Vitesse d'apprentissage

Nos résultats indiquent que l'architecture CRNN semble converger plus rapidement que l'architecture CFTNN. Les deux architectures nécessitent un temps proche s'entraîner, avec l'architecture CRNN légèrement (14%) plus rapide que l'architecture CFTNN. La courbe présentée dans la figure 10.2 montre une convergence plus rapide de l'architecture CRNN en début d'apprentissage, tandis que l'architecture CFTNN nécessite un peu plus de temps pour converger. De plus, l'architecture CFTNN parvient à des taux d'erreur plus faibles en validation, mais pourtant moins bons que ceux de l'architecture CRNN lors de la phase de test, indiquant ainsi que l'architecture CRNN semble mieux généraliser. Ces résultats peuvent notamment s'expliquer par les nombres de poids, l'architecture CRNN utilisant 1,7M paramètres et l'architecture CFTNN 3,5M.

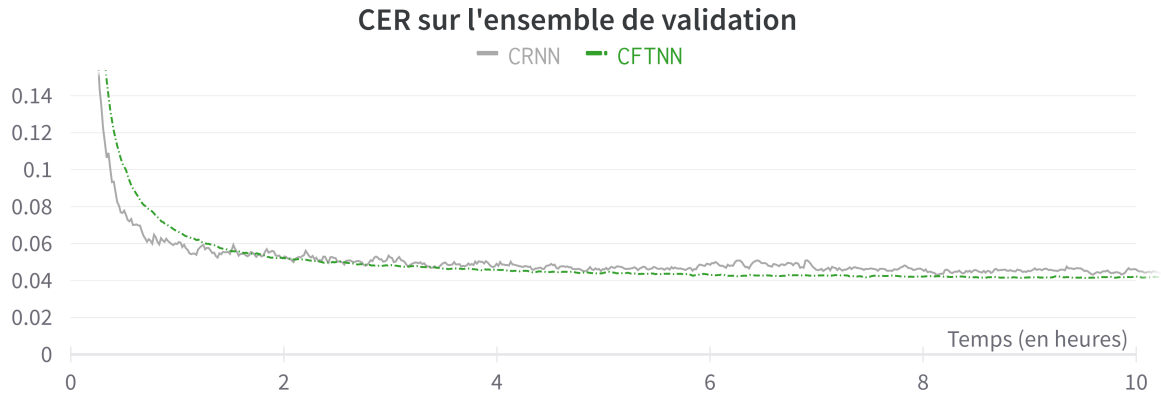


FIGURE 10.2 – Comparaison de l'évolution des entraînements des modèles CRNN et CFTNN. Ici, le CER en validation sur IAM est évalué en fonction du temps d'entraînement. Le CER est ici exprimé sous la forme d'une probabilité d'erreur (un CER de 0,1 correspond ainsi à 10% d'erreurs). Les modèles sont entraînés dans les mêmes conditions, en utilisant des données synthétiques et 4 NVIDIA Tesla V100.

Vitesse de prédiction Les mesures réalisées indiquent que l'architecture CRNN est plus rapide pour fournir un résultat (25ms) en comparaison à l'architecture CFTNN (29ms). Ce résultat est surprenant alors que les couches d'encodeur Transformers utilisées dans l'architecture CFTNN permettent un traitement non-séquentiel, et donc théoriquement plus rapide de la séquence. L'étude réalisée par Diaz et al. [Dia+21] indique, de même, des temps de prédiction relativement proches entre leur architecture d'encodeur Transformer et une architecture récurrente. Cependant, ils observent une vitesse de prédiction légèrement meilleure en utilisant leur architecture d'encodeur Transformer.

Pour tenter d'expliquer ce résultat, nous avons alors mesuré les temps de prédiction des couches récurrentes ainsi que des couches d'encodeur Transformer dans différents scénarios. Dans ces expériences, nous faisons varier la dimension D du modèle, c'est-à-dire le nombre de neurones par couche, ainsi que la taille de la séquence en entrée. Les modèles comparés sont composés uniquement de 4 couches de LSTM bidirectionnels (BLSTM), et de 4 blocs d'encodeur Transformer. Les résultats obtenus sont disponibles dans la table 10.3.

TABLE 10.3 – Comparaison des moyennes des temps de prédiction (en ms) entre des couches de BLSTM et des blocs d'encodeur Transformer. Ici, la moyenne des temps est calculée sur la base de 64 *batchs* de 64 vecteurs aléatoires.

Type de couches	D	longueur de séquence				
		128	256	512	1 024	2 048
BLSTM	256	**0,40**	**0,50**	**0,71**	**1,23**	2,01
	512	10,23	20,77	79,76	159,23	317,31
	1 024	7,96	31,16	158,13	334,42	697,22
Encodeur Transformer	256	1,02	1,09	1,25	1,29	**1,32**
	512	**1,07**	**1,23**	**1,28**	**1,29**	**1,36**
	1 024	**1,28**	**1,31**	**1,33**	**1,32**	**1,46**

Nous observons que les couches récurrentes sont plus rapides que les couches de Transformer dans des cas où la dimension D est faible (256 ou moins) et lorsque la taille des séquences n'est pas trop grande (1 024 ou moins). Cet écart lors de l'utilisation de faibles dimensions est potentiellement dû à des optimisations des LSTM dans le code de Pytorch. En effet, nous observons des écarts de temps conséquents entre une dimension D de 256 et de 512, alors que les temps devraient théoriquement rester inchangés (ou faiblement affectés). De plus, cela peut être lié à un nombre plus importants de couches successives,

de poids entraînables, ainsi que de traitement réalisé par des couches de Transformer, qui seraient alors plus lentes dans ces configurations.

Il est à noter en revanche que les couches d'encodeur Transformer restent très peu affectées par l'augmentation de la dimension D ainsi que de la longueur de la séquence. Ces couches permettent en effet un calcul en temps constant peu importe la longueur des séquences. Les couches récurrentes prennent pour leur part un temps croissant linéairement avec la longueur de la séquence. Dans le cas d'architectures larges et traitant de séquence de tailles importantes, il est alors bien plus efficace d'entraîner des couches de Transformer. Le gain en vitesse peut par exemple atteindre un facteur proche de 500, dans le cas $D = 1\ 024$ et avec des séquences de taille 2 048.

10.2.2 Intérêt d'un encodeur Transformer

Bien que notre architecture CFTNN soit plus lente que l'architecture CRNN du fait de la faible dimension des représentations internes, les deux architectures restent comparables. Nous étudions maintenant l'intérêt des deux architectures combinées avec un décodeur Transformer. Pour cela, nous comparons les résultats de l'architecture VLT obtenus avec un encodeur CFTNN (cas de base), ou avec un encodeur CRNN comme présenté dans l'architecture suivante :

VLT-CRNN L'architecture VLT-CRNN est similaire à l'architecture VLT, à l'exception de l'encodeur. L'encodeur est basé sur l'architecture CRNN au lieu de l'architecture CFTNN. La base convolutive reste la même, tandis que les 4 blocs d'encodeur Transformer sont remplacés par 4 couches de BLSTM. Le décodeur reste inchangé, avec 2 blocs de décodeur Transformer.

TABLE 10.4 – Résultats obtenus par notre architecture VLT-CRNN et VLT avec comme encodeur les architectures CRNN et CFTNN respectivement. Les résultats ont été obtenus sur le jeu de données IAM [MB02].

Architecture	Encodeur	IAM		IAM + données synthétiques	
		CER (%)	WER (%)	CER (%)	WER (%)
VLT-CRNN	CRNN	7,14	21,05	5,18	16,21
VLT	CFTNN	**5,81**	**18,91**	**4,40**	**14,77**

Dans la table 10.4, nous comparons ainsi les résultats obtenus en utilisant soit l'architecture CRNN soit l'architecture CFTNN.

L'architecture VLT obtient de bien meilleurs résultats avec un encodeur basé sur des couches Transformer plutôt que des couches récurrentes. Un encodeur Transformer semble plus efficace pour fournir un contexte pertinent à un décodeur lui aussi basé sur des couches Transformer.

10.2.3 Conclusion sur l'architecture CFTNN

Dans cette section, nous avons montré que l'architecture CFTNN est capable d'obtenir des résultats au niveau de ceux d'une architecture CRNN. Bien que sa configuration ne lui offre pas une meilleure vitesse, sa vitesse de prédiction reste néanmoins largement acceptable dans le cadre d'applications où un faible temps de prédiction est requis. L'architecture CFTNN reste légère et nécessite peu de place en mémoire. De plus, elle peut être entraînée efficacement avec peu de données annotées.

L'architecture CFTNN offre un excellent contexte pour des décodeurs Transformers, qui, nous le verrons, permettent d'offrir une forte réduction des taux d'erreur, notamment grâce à une modélisation efficace de la langue.

10.3 Bénéfices d'utiliser des architectures de Transformer autorégressives

Dans cette section, nous étudions les performances des architectures LT et VLT, proposées en tant que contribution de la thèse dans le chapitre 7. Nous les comparons avec les architectures CRNN, CFTNN, MT pour se comparer avec une architecture Transformer plus large, ainsi que l'architecture seq2seq qui est une architecture encodeur-décodeur séquence à séquence récurrente.

Dans les sections suivantes, l'essentiel des comparaisons est réalisé sur les résultats présentés dans la table 10.5. En particulier, nous étudions l'impact sur les taux d'erreur de l'ajout d'un décodeur Transformer (section 10.3.1), l'intérêt des architectures légères avec peu de données annotées (section 10.3.2), les temps de traitement des architectures (section 10.3.3), ainsi que les variances de nos différents modèles (section 10.3.4).

TABLE 10.5 – Comparaison des résultats obtenus par nos différentes architectures sur le jeu de données IAM. Les résultats sont donnés avec et sans l'utilisation de données synthétiques. Les architectures VLT et LT montrent d'excellents résultats.

Architecture	# param.	IAM		IAM + données synthétiques	
		CER (%)	WER (%)	CER (%)	WER (%)
CRNN	1,7M	6,35	22,93	5,78	20,90
CFTNN	3,5M	6,41	23,88	5,70	21,72
seq2seq	2,3M	8,87	30,31	7,02	24,62
VLT	5,6M	5,81	18,91	**4,40**	**14,77**
LT	7,7M	**5,69**	**18,29**	4,44	14,91
MT	29,9M	6,72	20,55	4,55	15,38

10.3.1 Impact d'un décodeur Transformer

Intérêt d'un décodeur Transformer

Nous démontrons tout d'abord l'intérêt d'utiliser un décodeur Transformer dans nos architectures. Comparées aux architectures CRNN et CFTNN, les architectures VLT et LT obtiennent de bien meilleurs résultats (table 10.5). Ces architectures qui utilisent un décodeur Transformer supplémentaire (VLT et LT), permettent une réduction relative du CER de 10% en moyenne et sans l'utilisation de données synthétiques. (Par réduction relative du CER, nous entendons qu'une réduction relative de 5% permet de passer d'un CER de 10% à 9,5%, ou encore de 5% à 4,75%.) Avec des données synthétiques, cet écart est bien plus prononcé, offrant une réduction du CER relative de 23% en moyenne. Ce résultat peut s'expliquer par la capacité de modélisation de la langue des décodeurs Transformer qui aident à réduire les erreurs commises par l'encodeur CFTNN. De plus, un décodeur Transformer montre qu'il bénéficie bien plus de l'utilisation de données synthétiques et d'un contenu textuel enrichi. Cela aide ainsi le modèle à généraliser sur des mots ou formulations non vus à l'entraînement.

Comparaison avec un décodeur récurrent

Comparées à l'architecture avec un décodeur récurrent (seq2seq), nos architectures obtiennent de meilleurs résultats, et en particulier avec des données synthétiques. Cela indique notamment qu'un décodeur Transformer semble capable d'une meilleure capacité à corriger des erreurs, et donc à modéliser la langue. En revanche, il est à noter que l'architecture seq2seq obtient de moins bons résultats que l'architecture CRNN, ce qui peut

indiquer que le décodeur tel quel n'est peut-être pas adapté pour la tâche. En comparaison avec nos architectures, l'architecture seq2seq utilise un nombre moins important de poids pour la partie décodeur.

Impact de la complexité d'un décodeur Transformer

Si l'on compare ensemble les résultats obtenus par nos architectures LT et VLT, nous remarquons que sans utiliser de données synthétiques, LT obtient des taux d'erreur plus faibles que VLT. À l'inverse, en utilisant des données synthétiques, VLT obtient de meilleurs résultats que LT. Selon nous, l'apprentissage d'une représentation à la fois compacte et pertinente de la langue est complexe à obtenir pour l'architecture VLT entraînée avec peu de données annotées. Avec un nombre de données annotées suffisant et éventuellement des données synthétiques, l'architecture VLT peut alors apprendre une modélisation plus compacte de la langue et obtenir de bons résultats. De plus, l'architecture possède une meilleure capacité de généralisation là où l'architecture LT est potentiellement plus sujette au surapprentissage. Nous verrons dans le chapitre 11, que ce dernier résultat s'observe particulièrement pour des entraînements réalisés sur des documents anciens. Les résultats obtenus avec l'architecture MT sont quant à eux abordés dans la section suivante.

10.3.2 Intérêt d'utiliser des architectures légères

Alors que les récentes tendances proposent des architectures de Transformer de plus en plus larges, à l'inverse, les architectures LT et VLT sont relativement légères. Les résultats de la table 10.5 montrent que, comparées à l'architecture MT qui est plus large, nos architectures VLT et LT obtiennent de meilleurs résultats. Sans données synthétiques, LT obtient un CER de 5,69% et VLT de 5,81%, ce qui est bien meilleur que le CER de 6,72% obtenu par MT. En s'entraînant avec des données synthétiques, l'écart est alors beaucoup plus faible, mais nos architectures légères obtiennent de légèrement meilleurs résultats.

Cette différence s'explique de manière logique par la différence de nombre de poids entre les architectures. L'architecture MT utilise en effet un total de 29,9M de poids, tandis que les architectures LT et VLT en utilisent 7,7M et 5,6M respectivement. Ce nombre de poids affecte directement la capacité des architectures à s'entraîner avec différentes quantités de données annotées. En effet, il est plus difficile d'entraîner une architecture large avec peu de données annotées. Kang et al [Kan+22] obtiennent un résultat similaire

avec l'architecture de Transformer large à 100M de paramètres qu'ils proposent. Celle-ci obtient alors un CER de 7,62% sans l'utilisation de données synthétiques, et de 4,67% grâce à un entraînement avec des données synthétiques.

De plus, nous avons observé lors des différentes expériences réalisées que l'architecture MT, qui est plus large, peut ne pas converger lorsque le nombre de données annotées est relativement faible. Nous n'observons en revanche pas ce phénomène avec les architectures légères proposées et les quantités de données étudiées.

Le choix entre architectures légères et larges est fortement impacté par le nombre de données annotées disponible. Les architectures larges demandent, en comparaison à des architectures légères, bien plus de données annotées pour être entraîné efficacement. Il est alors important de trouver un bon compromis entre données annotées et capacité de l'architecture. Néanmoins, les résultats obtenus démontrent que nos architectures légères sont capables d'obtenir de meilleurs résultats que des architectures larges sur le jeu de données IAM, qu'elles soient entraînées ou non avec des données synthétiques.

10.3.3 Vitesses d'entraînement et de prédiction

Nous proposons maintenant d'évaluer les vitesses d'entraînement ainsi que le temps de prédiction pris par nos architectures de Transformer autorégressives. Contrairement à l'architecture CFTNN, nous n'utilisons pas la version des Transformers implémentée dans la librairie Pytorch, puisque nous avons observé de meilleurs temps en prédiction avec notre propre version de l'architecture Transformer. Néanmoins, les résultats entre ces deux versions restent relativement proches au niveau des taux d'erreur ou du temps requis. Il est à noter que certaines optimisations supplémentaires peuvent être réalisées dans le but de fournir des temps plus faibles. La table 10.6 présente les résultats obtenus par les différentes architectures comparées.

Vitesse d'entraînement

Les architectures VLT et LT, s'entraînent globalement assez vite : 42h pour VLT et 46h pour LT. L'architecture MT s'entraîne en seulement 46h malgré un nombre de poids entraînables plus important. Cependant, il est à noter que l'architecture MT utilise un nombre de couches similaire à l'architecture LT. Seul le nombre de neurones par couche est modifié, ce qui n'influence pas, ou peu, le temps pour compléter une époque. En revanche, il serait attendu que l'architecture MT nécessite plus d'époques pour converger.

TABLE 10.6 – Évaluation des vitesses de nos architectures sur le jeu de données IAM, entraînées et évaluées avec quatre cartes graphiques NVIDIA Tesla V100. Le temps d'entraînement correspond à la durée d'entraînement moyenne observé sur plusieurs entraînements réalisés avec des données réelles et synthétiques. Les statistiques d'évaluation sont mesurées sur l'ensemble des *batchs* lors de l'étape de prédiction, avec une taille de *batch* de 64.

Architecture	# param	Temps d'entraînement (en h)	Temps pour évaluer un *batch* (en s)
CRNN	1,7M	19	0,025 ± 0,005
CFTNN	3,5M	22	0,029 ± 0,004
seq2seq	2,2M	37	0,136 ± 0,016
VLT	5,6M	42	1,000 ± 0,042
LT	7,7M	46	1,763 ± 0,066
MT	29,9M	46	1,804 ± 0,086

Selon nous, l'architecture MT n'est pas entraînée avec suffisamment de données annotées, ce qui donne lieu à une convergence sous optimale, ou à du surapprentissage. Elle ne modélise alors pas aussi bien l'information comparée à une architecture plus légère telle que VLT.

Les architectures VLT et LT nécessitent des temps d'entraînement plus longs que des architectures plus simples telles que les architectures CRNN et CFTNN. Néanmoins, la figure 10.3 montre que pour un même taux d'erreur, nos architectures VLT et LT atteignent ces performances plus vite que les architectures CRNN ou CFTNN.

Vitesse de prédiction

Les architectures LT et VLT sont bien plus lentes en comparaison à l'architecture CFTNN à cause d'un décodage séquentiel durant la phase de prédiction. C'est aussi le cas des architectures CRNN et seq2seq, mais nous avons vu dans la section 10.2.1 que des couches récurrentes sont plus rapide dans ces configurations.

L'architecture VLT est bien plus rapide en comparaison à l'architecture LT puisqu'elle nécessite en moyenne 1,000 seconde contre 1,763 pour l'architecture LT. Cette différence s'explique notamment par le nombre de couches de décodeur Transformer. L'architecture LT possédant 4 couches de décodeur contre 2 pour VLT, une itération du décodeur est alors globalement 2 fois plus longue. Les architectures LT et MT ont quant à elles des temps de prédiction proches, malgré un nombre de poids plus important pour l'architecture

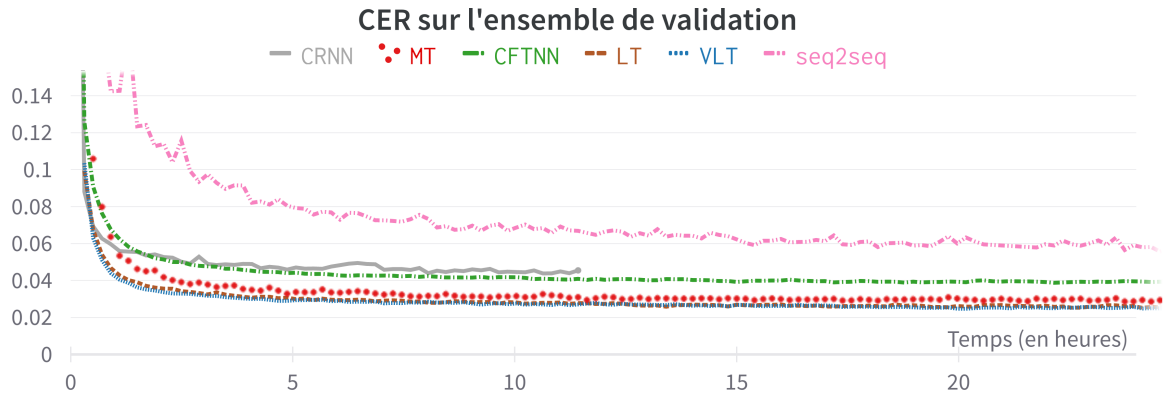


FIGURE 10.3 – Comparaison de l'évolution des entraînements de différents modèles. Ici, le CER en validation sur IAM est évalué en fonction du temps d'entraînement. Le CER est exprimé sous la forme d'une probabilité d'erreur. Les modèles sont entraînés avec des données synthétiques, et dans des conditions similaires, en utilisant 4 NVIDIA Tesla V100.

MT. Elles possèdent en effet toutes deux le même nombre de couches, mais un nombre de neurones par couche différent. Cela n'affecte pas le temps, comme démontré par la table 10.3 (p. 161).

Les architectures VLT et LT sont bien moins rapides en prédiction en comparaison à des architectures plus simples telles que l'architecture CRNN ou l'architecture CFTNN. Néanmoins, elles sont capables de fournir une reconnaissance des caractères bien plus fiable. Selon le cadre applicatif, et si la vitesse de prédiction est un facteur important, il convient alors de choisir une architecture qui offre un bon compromis.

10.3.4 Étude de la variance de nos modèles

Nous proposons maintenant d'évaluer la variance de nos modèles, et en particulier nous comparons celles obtenues par les architectures VLT et LT avec une architecture CFTNN. Pour cela, nous entraînons 5 modèles initialisés aléatoirement pour chacune des architectures sur le jeu de données IAM, tout en considérant l'utilisation de données synthétiques. La table 10.7 présente les résultats obtenus.

En comparaison à l'architecture CFTNN, nos architectures VLT et LT obtiennent certes de meilleurs taux d'erreur en moyenne, mais des écarts-types observés plus importants (0,07 contre 0,19 et 0,15 pour le CER). Cela peut s'expliquer par la présence d'un décodeur Transformer qui est important pour l'apprentissage, mais qui augmente la com-

TABLE 10.7 – Moyennes et variances des taux d'erreur empiriques de nos architectures sur le jeu de données IAM. Pour chaque architecture, 5 modèles initialisés aléatoirement sont entraînés. Ces modèles sont entraînés en utilisant des données synthétiques.

Architectures	CFTNN		VLT		LT	
	CER (%)	WER (%)	CER (%)	WER (%)	CER (%)	WER (%)
Expériences	5,70	21,72	4,40	14,77	4,44	14,91
	5,71	21,72	4,50	15,06	4,50	15,23
	5,76	21,79	4,67	15,66	4,51	15,10
	5,76	21,81	4,78	15,54	4,68	15,42
	5,88	22,16	4,88	16,10	4,80	15,57
Moyenne ± écart-type	5,76 ± 0,07	21,84 ±0,18	4,68 ± 0,19	15,42 ± 0,52	4,59 ± 0,15	15,25 ± 0,26

plexité de l'architecture. L'entraînement est alors complexifié, ce qui résulte selon nous en des écarts-types accrus.

En moyenne, l'architecture LT obtient de plus faibles taux d'erreur que l'architecture VLT, avec un CER moyen de 4,59 ± 0,15 % contre 4,68 ± 0,19 %. De plus, nous observons moins de variance sur l'architecture LT que sur l'architecture VLT. En revanche, nous obtenons le CER le plus faible de 4,40% avec l'architecture VLT, tandis que l'architecture LT obtient au mieux un CER proche de 4,44%. Ces deux architectures obtiennent des résultats assez proches, bien que LT semble légèrement meilleur en moyenne dans le cadre de documents modernes. L'architecture LT a probablement un décodeur plus adapté à la modélisation d'informations contextuelles sur ce type de documents. L'architecture VLT utilise un décodeur deux fois plus léger, qui peut parfois être insuffisant pour modéliser correctement la langue dans ces documents.

10.3.5 Bilan sur les architectures de Transformer autorégressives

Dans cette section, nous avons démontré que les architectures de Transformer autorégressives sont capables d'obtenir d'excellents résultats sur le jeu de données IAM. Elles restent légères, ce qui leur confère différents avantages sur des architectures plus larges, comme un entraînement plus rapide et plus efficace avec peu de données. De plus, elles sont capables d'obtenir de meilleurs résultats que des architectures de Transformer plus larges, en utilisant uniquement les données d'entraînement, ou en considérant aussi des données synthétiques. En particulier, l'architecture VLT obtient un CER de 5,81% sans

données synthétiques, et un excellent taux CER de 4,40% lorsqu'elle est entraînée avec des données synthétiques.

Ces architectures de Transformer restent quand bien même complexes à entraîner, et nécessitent l'utilisation de diverses stratégies pour être entraînée efficacement.

10.4 Impact des données sur l'entraînement

Dans cette section, nous montrons l'impact des différentes stratégies appliquées lors de l'entraînement des architectures proposées dans ce manuscrit. Ces stratégies sont d'autant plus importantes qu'elles permettent d'améliorer les performances des architectures, ainsi que la stabilité de l'entraînement. C'est en particulier le cas lors d'entraînement avec peu de données annotées.

Dans un premier temps, la section 10.4.1 évalue l'impact des différents traitements appliqués sur les données d'entraînement. Nous évaluons ensuite l'impact d'une fonction de coût hybride sur l'entraînement dans la section 10.4.2, tandis que la section 10.4.3 présentera une manière d'utiliser habilement l'encodeur de nos architectures. La section 10.4.4 présente quant à elle l'impact de l'ajout d'information positionnelle supplémentaire sur la séquence produite par l'encodeur de notre architecture.

10.4.1 Impact des traitements appliqués sur les données d'entraînement

Génération de données synthétiques

Comme nous l'avons vu dans les approches de l'état de l'art, les architectures de Transformer, et en particulier les plus larges, utilisent des données synthétiques pour être entraînées efficacement. Dans les sections précédentes, nous avons par ailleurs étudié l'impact de différents choix d'architectures, que ce soit lors d'entraînement sans données synthétiques ou en utilisant des données synthétiques. Dans cette partie, nous concentrons la discussion sur l'intérêt d'utiliser des données synthétiques. En particulier, nous discutons en détail de l'impact pour les différentes architectures proposées. Nous reprenons pour cela les résultats précédents qui ont été présentés dans la table 10.5 (p. 164).

Nous observons une réduction notable des taux d'erreur, et ce, sur l'ensemble de nos architectures. Par exemple, notre architecture CFTNN, entraînée avec des données synthétiques, obtient un CER de 5,70% au lieu de 6,41%, soit un gain relatif de 10%.

Nous notons cependant que les architectures qui utilisent un décodeur Transformer bénéficient plus de l'apport de données synthétiques que les architectures plus simples CRNN ou CFTNN. L'architecture VLT passe quant à elle d'un CER de 5,81% à 4,40% grâce à l'apport de données synthétiques lors de l'entraînement. Cela représente un gain bien plus important de 24% relativement. Le WER suit quant à lui une réduction similaire au CER. Cette différence notable entre architectures peut s'expliquer par les bonnes capacités de modélisation de la langue des couches de décodeur Transformer. L'utilisation de données synthétiques lors de l'entraînement apporte ainsi un contenu textuel enrichi, ce qui bénéficie particulièrement à ces architectures pour apprendre une bien meilleure modélisation de la langue.

Néanmoins, nous tenons à souligner le fait que nos architectures, de par leur faible nombre de paramètres (ainsi que de diverses autres stratégies), peuvent quand même être entraînées efficacement sans l'utilisation de données synthétiques. En comparaison avec l'architecture MT, plus large, qui parvient à un CER de 6,72%, nos architectures légères VLT et LT obtiennent des CER de 5,81% et 5,69% respectivement. Les architectures de Transformer larges sont, en effet, bien plus complexes à entraîner, et nécessitent l'utilisation de données synthétiques pour obtenir de bons résultats [Kan+22].

Nos résultats montrent qu'une architecture de Transformer large n'est pas forcément nécessaire pour obtenir de bons résultats, et que des architectures de Transformer légères sont capables d'obtenir de bons résultats. Sans données synthétiques, nos architectures légères s'entraînent efficacement, contrairement aux architectures larges. Les architectures légères VLT et LT bénéficient tout de même d'un important gain avec des données synthétiques.

Augmentations de données

Nous évaluons maintenant l'impact d'augmentations sur les données réelles. En particulier, il nous semble intéressant d'étudier si les données synthétiques peuvent se substituer à l'augmentation de données. Pour cela, nous présentons dans la table 10.8 des résultats obtenus par l'architecture VLT entraînée avec ou sans augmentations de données.

Les résultats montrent qu'il est important d'utiliser de l'augmentation de données sur les données réelles. Sans utiliser de données synthétiques lors de l'entraînement, l'augmentation de données se révèle être très efficace, comme attendu. Cela permet une amélioration du CER de 9,88% à 5,81%, soit une réduction relative de 41%. En combinaison avec des données synthétiques (elles aussi augmentées), l'utilisation d'augmentation de données

TABLE 10.8 – Comparaison des taux d'erreur obtenus par notre architecture VLT sur le jeu de données IAM, avec ou sans l'utilisation d'augmentations de données sur les données réelles.

Augmentation de données	IAM		IAM + données synthétiques	
	CER (%)	WER (%)	CER (%)	WER (%)
Non	9,88	28,79	6,46	21,23
Oui	**5,81**	**18,91**	**4,40**	**14,77**

réelles permet de réduire le CER de 6,46% à 4,40%. Cela représente une amélioration tout de même importante de 32% relativement.

Cet écart important entre les deux scénarios, à savoir avec ou sans l'utilisation de données synthétiques permet de démontrer l'importance cruciale de l'utilisation de données réelles diverses et annotées. En effet, l'entraînement de notre architecture dépend fortement de l'utilisation de ces données réelles dont la diversité en entrée est obtenue par des stratégies d'augmentation. En particulier, au vu des taux d'erreur obtenus, nous notons que le gain est plus significatif avec de l'augmentation de données que d'utiliser uniquement des données synthétiques.

10.4.2 Intérêt d'une fonction de coût hybride

Nous montrons maintenant l'intérêt d'entraîner nos architectures avec une fonction de coût hybride, c'est-à-dire reposant sur l'ajout d'une fonction de coût issue de l'approche de classification temporelle connexioniste [Gra+06] (CTC) en plus de l'entropie croisée (CE).

TABLE 10.9 – Impact de l'utilisation d'une fonction de coût hybride sur le jeu de données IAM. Dans cette expérience, nous comparons les architectures LT et VLT entraînées avec une fonction de coût de type entropie croisée seule (CE seule) ou hybride (CTC + CE). Ces architectures sont entraînées avec ou sans données synthétiques.

Fonction de coût	Architecture	IAM		IAM + données synthétiques	
		CER (%)	WER (%)	CER (%)	WER (%)
CE seule	VLT	14,84	31,99	6,63	18,35
	LT	17,09	33,62	8,53	20,88
hybride (CTC + CE)	VLT	5,81	18,91	**4,40**	**14,77**
	LT	**5,69**	**18,29**	4,44	14,91

Les résultats présentés dans la table 10.9 indiquent que l'utilisation d'une fonction de coût hybride permet d'entraîner efficacement les architectures VLT et LT proposées.

En particulier, l'ajout d'une fonction de coût CTC se révèle être crucial lors d'entraînements réalisés avec peu de données annotées, c'est-à-dire sans l'ajout de données synthétiques. Nous observons une diminution drastique du CER de 14,84% à 5,81% et de 17,09% à 5,69% pour les architectures VLT et LT respectivement. Cette différence s'explique par le fait que l'ajout d'une fonction de coût CTC en sortie de l'encodeur permet d'entraîner rapidement et efficacement la partie encodeur, et donc les couches les plus profondes de l'architecture. Bien que des couches résiduelles sont utilisées dans les différentes couches de Transformer et que nous utilisons des architectures peu profondes comparées à d'autres, cette fonction de coût additionnelle offre un second gradient pertinent pour l'entraînement de ces couches. De plus, comme le décodeur pondère les caractéristiques produites par l'encodeur pour produire la sortie du décodeur, la fonction de coût CTC bénéficie aussi à l'entraînement de la partie décodeur en fournissant des caractéristiques pertinentes tôt lors de l'entraînement. La fonction de coût hybride permet ainsi de particulièrement aider la convergence des architectures Transformer. La figure 10.4 illustre cette différence.

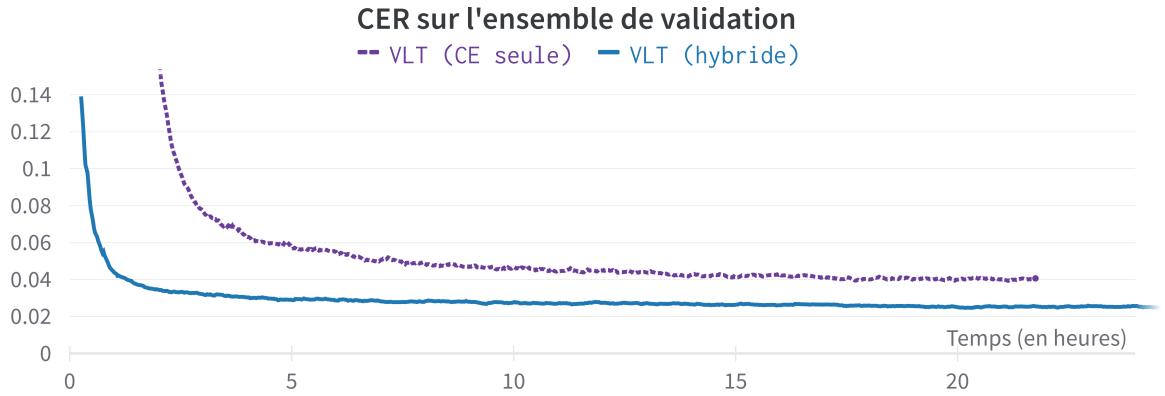


FIGURE 10.4 – Évolution du CER en validation sur IAM en utilisant ou non une fonction de coût hybride pour l'architecture VLT entraînée avec des données synthétiques. Le CER est ici exprimé sous la forme d'une probabilité d'erreur. Les modèles sont tout deux entraînés avec les mêmes autres paramètres, et dans des conditions similaires, en utilisant 4 NVIDIA Tesla V100.

Avec des données synthétiques, la réduction des taux d'erreur est plus faible, mais reste cependant importante puisqu'elle permet une réduction du CER d'environ 33% relativement pour l'architecture VLT.

10.4.3 Impact du décodeur sur l'encodeur

La fonction de coût hybride a montré son intérêt pour les architectures VLT et LT. Lors d'un tel entraînement, l'encodeur reçoit alors des gradients provenant à la fois de l'approche CTC, mais aussi de la partie décodeur. Il peut alors être intéressant de comparer les performances d'un encodeur entraîné de la sorte avec celles de l'architecture CFTNN (le même encodeur entraîné seul). Nous proposons ainsi d'évaluer l'architecture suivante :

Encodeur VLT L'architecture Encodeur VLT désigne l'encodeur de l'architecture VLT entraînée avec une fonction de coût hybride. La structure de l'architecture Encodeur VLT est exactement la même que l'architecture CFTNN, cependant, elle est entraînée de manière conjointe avec un décodeur Transformer.

TABLE 10.10 – Comparaison des architectures CFTNN et Encodeur VLT. CFTNN est entraînée de manière isolée, tandis que l'architecture Encodeur VLT correspond à l'architecture CFTNN entraînée de manière conjointe.

Architecture	IAM		IAM + données synthétiques	
	CER (%)	WER (%)	CER (%)	WER (%)
CFTNN	6,41	23,88	5,70	21,72
Encodeur VLT	**5,96**	**22,89**	**5,56**	**21,42**

Les résultats de la table 10.10 montrent que l'entraînement conjoint bénéficie à l'encodeur. Une réduction des taux d'erreur peut en effet être observée, de 8% relativement sans donnés synthétiques, et de 3% relativement en utilisant des données synthétiques. Ce résultat a par ailleurs aussi été observé dans l'approche séquence à séquence proposée par Michael et al [Mic+19].

L'architecture Encodeur VLT est donc intéressante et permet, grâce à un entraînement conjoint, d'obtenir une version améliorée de l'architecture CFTNN. Tout comme cette dernière, l'architecture Encodeur VLT a une bonne vitesse de prédiction, et peut donc être utilisée dans le cas où la vitesse de prédiction serait un critère important. Comparée à l'architecture VLT, l'architecture Encodeur VLT est 30 fois plus rapide en prédiction pour une augmentation du taux d'erreur de 26% relativement. L'architecture Encodeur VLT peut par exemple être utilisée pour fournir jusqu'à 2 000 prédictions sur des lignes de texte par seconde, soit environ l'équivalent de 60 pages de lignes de texte par seconde.

Cependant, il est à noter qu'un entraînement conjoint nécessite en contrepartie un temps d'entraînement plus élevé (42 h au lieu de 22 h dans notre cas), ainsi qu'une quantité de données annotées suffisante pour entraîner le décodeur en plus de l'encodeur.

10.4.4 Ajout d'informations positionnelles

Nous montrons maintenant l'intérêt d'ajouter de l'information positionnelle sur la séquence de sortie de l'encodeur. Pour cela, nous évaluons les résultats obtenus lors de l'entraînement de notre architecture VLT, entraînée avec ou sans l'ajout d'une couche d'encodage positionnelle sur la sortie de l'encodeur.

TABLE 10.11 – Comparaison des taux d'erreur de l'architecture VLT avec ou sans l'ajout d'informations positionnelles entre l'encodeur et le décodeur.

Encodage positionnel supplémentaire	IAM		IAM + données synthétiques	
	CER (%)	WER (%)	CER (%)	WER (%)
Non	7,54	21,60	4,71	15,55
Oui	**5,81**	**18,91**	**4,40**	**14,77**

Les résultats de la table 10.11 montrent ainsi que l'ajout d'informations positionnelles bénéficie à notre architecture VLT. Dans un entraînement sans données synthétiques, et donc avec un nombre assez restreint de données annotées, nous observons un gain important dans les performances de l'architecture VLT. Le CER est alors réduit de 7,54% à 5,81%, ce qui représente une réduction du CER relative de 23%. Cependant, ce gain est plus nuancé lors d'un entraînement avec des données synthétiques, puisqu'il permet une réduction du CER de 4,71% à 4,40% (6% relativement).

Bien que de l'information positionnelle soit ajoutée entre les couches convolutives et les couches d'encodeur Transformer, nous pensons qu'ajouter à nouveau de l'information positionnelle permet à l'architecture d'apprendre plus facilement. Le décodeur accède ainsi, grâce à la séquence produite par l'encodeur, à de l'information sur les caractères, ainsi que leur position de manière plus claire. La figure 10.5 (p. 176) montre que cela n'affecte cependant pas la vitesse de convergence. Les modèles se différencient en revanche par leurs taux d'erreur sur la fin des entraînements.

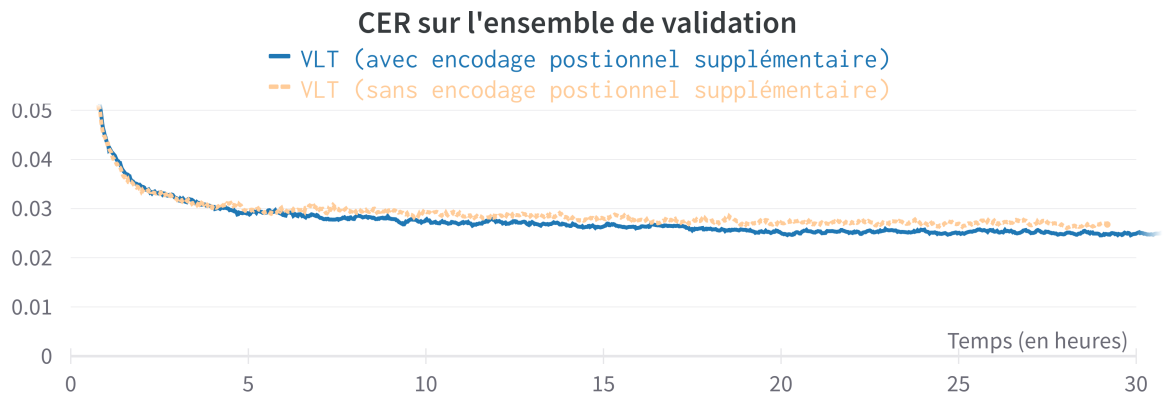


FIGURE 10.5 – Effets d'utiliser un encodage positionnel supplémentaire sur le CER en validation (exprimé sous la forme d'une probabilité d'erreur). Les modèles VLT sont tout deux entraînés avec les mêmes autres paramètres, avec des données synthétiques, en utilisant 4 NVIDIA Tesla V100.

10.4.5 Conclusion sur les différentes stratégies proposées

Nous avons évalué l'impact des différentes stratégies sur l'entraînement de nos architectures, et avons montré l'importance de celles-ci.

Du fait que nous utilisons des architectures complexes basées sur des couches de Transformer, la variété des données est un facteur clé pour entraîner efficacement les architectures proposées. L'utilisation d'augmentation de données ainsi que la génération de données synthétiques ont alors un impact fort, comme démontré dans la section 10.4.1.

L'utilisation d'une fonction de coût hybride combinant une fonction de coût CTC pour entraîner l'encodeur ainsi qu'une fonction de coût CE permet aussi de grandement aider à l'apprentissage. Grâce à celle-ci, nos modèles convergent plus vite, et sont capables d'obtenir de bien meilleurs résultats.

Enfin, nous avons aussi montré que l'utilisation d'un encodage positionnel placé entre l'encodeur et le décodeur permet une réduction supplémentaire des taux d'erreur.

10.5 Entraînements avec peu de données annotées

Nous évaluons maintenant les performances de nos architectures entraînées sur différents scénarios avec peu de données annotées. Cela nous permet de valider que les différentes stratégies que nous avons introduites permettent de conserver de bonnes performances malgré le faible nombre de données.

À cette fin, nous proposons de diviser le jeu de données IAM en 5 sous-ensembles contenant respectivement 10, 25, 50, 75 et 100% des lignes de textes du jeu de données complet. Les sous-ensembles d'entraînement, et de validation, sont obtenus en sélectionnant des images de lignes de texte aléatoirement parmi les données d'entraînement, et respectivement de validation, du jeu de données complet. De plus, nous faisons en sorte que l'intégralité des exemples d'un sous-ensemble soit contenue dans les sous-ensembles contenant plus de données. La table 10.12 résume la composition de ces différents sous-ensembles. L'ensemble de test reste quant à lui inchangé, et tous nos modèles sont donc évalués sur les mêmes données.

TABLE 10.12 – Caractéristiques des sous-jeux de données d'IAM que nous proposons pour évaluer les performances de nos architectures avec un nombre réduit de données annotées.

Sous-ensemble	# lignes d'entraînement	# lignes validation
10%	648	98
25%	1 621	244
50%	3 241	488
75%	4 862	732
100%	6 482	976

Nous étudions les performances des architectures CRNN, CFTNN, VLT et LT entraînées avec peu de données réelles. Nous utilisons cependant de l'augmentation de données dans le but d'aider l'entraînement des architectures. De la même manière que nous adaptons les données d'entraînement réelles, nous proposons de garder un nombre de données synthétiques stable et indépendant du scénario étudié (10 000 par époque). La figure 10.6 présente les différents résultats obtenus dans ces scénarios.

Grâce à l'utilisation de données synthétiques, nous observons que les différentes architectures obtiennent des taux d'erreur globalement peu affectés par de faible quantité de données annotées. Elles obtiennent un CER acceptable [Str+18], à savoir inférieur à 10%, avec très peu de données annotées (entre 500 et 1 000). Sur le scénario utilisant 10% des données annotées d'IAM, seul l'architecture CFTNN n'obtient pas un seuil acceptable.

Avec peu de données annotées réelles, l'architecture CRNN semble plus adaptée que l'architecture CFTNN. Cependant, nous observons que l'écart entre les architectures diminue au fur et à mesure. En effet, l'architecture CFTNN obtient un CER meilleur que celui de l'architecture CRNN en utilisant le maximum de données annotées. Cela peut s'expliquer par le nombre de poids, ainsi que par l'utilisation de couches Transformer.

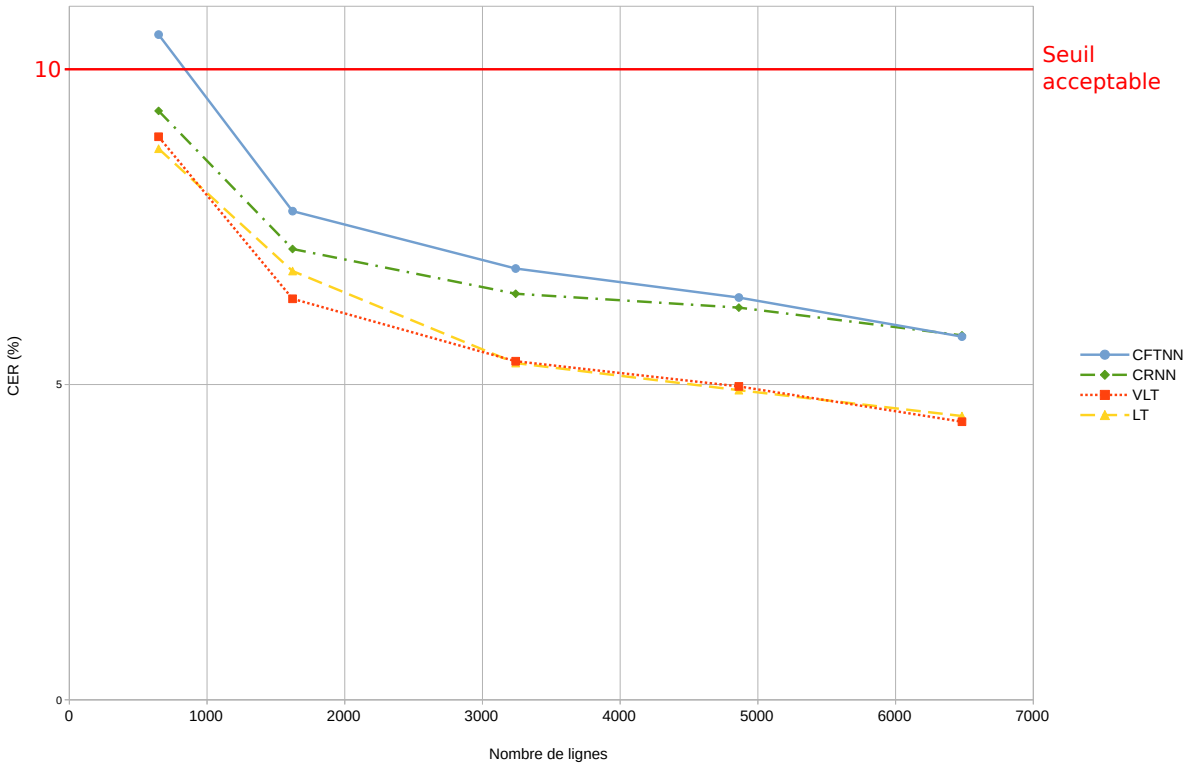


FIGURE 10.6 – Évaluation des performances de nos architectures entraînées avec différentes quantités de données annotées provenant du jeu de données IAM, ainsi qu'avec des données synthétiques.

Les architectures VLT et LT obtiennent quant à elles de meilleurs résultats que les architectures CFTNN et CRNN, et ce même dans les scénarios avec le moins de données annotées disponibles. L'apport de données synthétiques permettrait aux modèles d'apprendre efficacement à modéliser la langue, et donc à réduire le nombre d'erreurs.

Enfin, VLT et LT obtiennent globalement des résultats assez proches, avec un écart de plus en faible à mesure que le nombre de données annotées réelles augmente. De plus, nous notons que l'architecture VLT obtient de légèrement meilleurs résultats lorsque les données annotées réelles sont maximales.

10.6 Impact de la stratégie de prédiction

Dans cette section, nous évaluons l'intérêt de la stratégie de prédiction basée sur de l'augmentation en test et un algorithme de vote, proposée dans la section 9.2 (p. 143).

Pour cela, nous comparons les résultats obtenus par les architectures proposées avec ou sans utiliser la stratégie de prédiction.

De plus, nous comparons notre stratégie de prédiction à celle proposée par Soullard et al. [Sou+19], dont nous nous inspirons. Nous rappelons que comparée à l'approche de Soullard et al., notre stratégie inclut un filtrage des prédictions trop éloignées pour réduire l'impact des prédictions aberrantes et permettre l'utilisation d'un plus grand nombre d'images dupliquées.

Dans les résultats présentés, nous appliquons la stratégie de Soullard et al. avec 15 prédictions par image réelle, tandis que notre stratégie utilise 60 prédictions. L'utilisation d'un nombre de prédictions plus important pour la stratégie de Soullard et al. résulte en effet en des temps d'alignement élevés qui nous contraignent à utiliser un nombre de prédictions restreint.

TABLE 10.13 – Impact de la stratégie de prédiction sur le jeu de données IAM. Nous comparons pour chacune des architectures proposées : l'absence de stratégie de prédiction ; la stratégie proposée par Soullard et al. avec 15 prédictions ; ainsi que notre stratégie de prédiction avec soit 15, soit 60 prédictions.

Architectures	Stratégie de vote	IAM		IAM + synth.	
		CER (%)	WER (%)	CER (%)	WER (%)
CFTNN	-	6,41	23,88	5,76	21,81
	[Sou+19] (15)	6,20	23,06	5,61	21,44
	Notre (15)	6,15	22,93	5,56	21,33
	Notre (60)	**6,11**	**22,81**	**5,53**	**21,15**
VLT	-	5,81	18,91	4,40	14,77
	[Sou+19] (15)	5,71	18,90	4,25	14,62
	Notre (15)	5,68	**18,79**	4,25	**14,60**
	Notre (60)	**5,67**	18,80	**4,23**	14,61
LT	-	5,69	18,29	4,50	15,23
	[Sou+19] (15)	-	-	**4,32**	**14,72**
	Notre (15)	5,64	**18,22**	**4,32**	14,79
	Notre (60)	**5,61**	18,72	4,33	14,75

Dans les tables 10.13 et 10.14, nous présentons les résultats obtenus sur les jeus de données IAM et RIMES.

Les résultats indiquent une réduction notable des taux d'erreur obtenus par nos architectures de 2 à 8% relativement. Elle semble légèrement plus bénéfique lorsqu'elle est appliquée sans données synthétiques, ou elle apporte une réduction relative du CER de 6% au lieu de 5% avec des données synthétiques. De plus, l'architecture CFTNN semble

TABLE 10.14 – Impact de la stratégie de prédiction sur le jeu de données RIMES. Nous comparons pour chacune des architectures proposées : l'absence de stratégie de prédiction ; la stratégie proposée par Soullard et al. avec 15 prédictions ; ainsi que notre stratégie de prédiction avec soit 15, soit 60 prédictions.

Architectures	Stratégie de vote	RIMES		RIMES + synth.	
		CER (%)	WER (%)	CER (%)	WER (%)
CFTNN	-	3,71	13,14	3,33	11,71
	[Sou+19] (15)	3,52	12,62	3,30	11,52
	Notre (15)	3,47	12,34	3,22	11,18
	Notre (60)	**3,45**	**12,18**	**3,18**	**11,14**
VLT	-	4,01	10,59	3,02	8,85
	[Sou+19] (15)	3,71	10,58	2,92	8,78
	Notre (15)	3,70	10,45	**2,81**	8,51
	Notre (60)	**3,64**	**10,24**	2,82	**8,47**
LT	-	3,62	9,85	2,70	**7,14**
	[Sou+19] (15)	-	-	2,53	7,33
	Notre (15)	3,56	9,81	**2,47**	7,19
	Notre (60)	**3,49**	**9,76**	2,51	7,26

bénéficier d'une réduction du CER légèrement plus importante, probablement au nombre plus élevé d'erreurs commises par le modèle.

La stratégie de prédiction est particulièrement bénéfique dans les scénarios les plus sensibles aux variations dans les images, comme lors d'entraînement sans données synthétiques.

Comparée à la stratégie de prédiction proposée par Soullard et al., notre stratégie de prédiction, en utilisant 15 ou 60 prédictions, se révèle être plus efficace et permet une réduction relative du CER de 2% en moyenne. En revanche, il est à noter que l'approche proposée par Soullard et al. ne permet pas une convergence en un temps raisonnable lorsqu'elle est appliquée sur l'architecture LT entraînée sans données synthétiques. À cause d'un surapprentissage, celle-ci est beaucoup plus sensible aux variations dans les images, ce qui peut parfois résulter en des prédictions aberrantes. La stratégie proposée semble ainsi être plus robuste que celle proposée par Soullard et al., grâce à des taux d'erreur plus faibles, mais aussi à une meilleure stabilité. Elle permet en outre d'utiliser plus d'images dupliquées et donc plus de prédictions afin d'obtenir une prédiction finale consolidée, là ou l'approche de Soullard et al. ne permet pas d'utiliser un tel nombre en un temps raisonnable.

La stratégie de prédiction proposée permet, comme le démontrent les résultats, une réduction notable des taux d'erreur. Cependant, elle nécessite en contrepartie un temps de prédiction plus en fonction du nombre d'images dupliquées. Elle n'est donc pas recommandée dans les cas où la vitesse de prédiction est essentielle.

Nous remarquons enfin une faible différence au niveau des taux d'erreur obtenus avec notre stratégie et un nombre de prédiction de 15 ou de 60. 15 prédictions semblent suffisantes pour permettre une réduction significative des taux d'erreur, tout en ne nécessitant pas un temps trop important.

Par la suite, nous utilisons les résultats obtenus avec la stratégie de vote pour se comparer à l'état de l'art. Les différentes approches de l'état de l'art n'utilisent en revanche pas de stratégies de prédiction, à l'exception de l'approche présentée par Dutta et al. [Dut+18]. Il est à noter que la stratégie de vote s'applique à de tels systèmes et qu'ils peuvent en conséquent bénéficier d'une amélioration similaire.

10.7 Comparaison avec l'état de l'art

Dans cette section, nous comparons les architectures proposées, c'est-à-dire CFTNN, VLT et LT, avec les différentes approches de l'état de l'art. Nos architectures sont entraînées avec les différentes stratégies discutées dans ce chapitre. Les résultats seront donnés avec et sans données synthétiques dans le but de permettre des comparaisons plus équitables. En effet, toutes les approches n'utilisent pas de données synthétiques et pourraient bénéficier de meilleurs résultats avec leur utilisation. Nous rappelons en revanche, comme vu précédemment, que les architectures utilisant un décodeur Transformer bénéficient bien plus de l'apport de données synthétiques.

Nous nous évaluons sur le jeu de données IAM, mais aussi sur le jeu de données RIMES. Les résultats sont présentés dans la table 10.15. Il est cependant à noter que les découpages des données d'apprentissages en ensembles d'entraînements et de validation diffèrent d'une approche à une autre sur le jeu de données RIMES.

Comparées aux autres architectures Transformer de l'état de l'art, nos architectures de Transformer sont généralement plus légères, tout en étant capables d'obtenir de bons résultats. L'architecture CFTNN utilise 3,5M paramètres, soit 3 à 4 fois moins de paramètres que l'architecture "S-Attn + CTC" proposée par Diaz et al. [Dia+21]. L'architecture "CNN-SAN" proposée par D'Arce et al. [dAr+22] utilise deux fois moins de paramètres que la nôtre, cependant, notre architecture CFTNN obtient un CER bien meilleur de

TABLE 10.15 – Comparaison avec l'état de l'art sur les jeu de données IAM et RIMES. Les résultats en gras dénotent les meilleurs résultats de chaque catégorie, avec ou sans l'utilisation d'un modèle de langue (LM) externe. Nous indiquons aussi les approches qui utilisent des stratégies de prédiction (strat. pred.).

Type d'architecture	Méthode	LM ext.	strat. pred.	# param.	IAM	IAM + synth.	IAM + ext.	RIMES	RIMES + synth.	RIMES + ext.
					CER (%)	CER (%)	CER (%)	CER (%)	CER (%)	CER (%)
CRNN	GCRNN [BM17]	✓	-	725k	-	-	3,2	-	-	**1,9**
	CRNN [WZG22]	-	-	3,2M	5,47	4,99	-	5,31	4,25	-
	CRNN [Pui17]	-	-	-	5,8	-	-	**2,3**	-	-
	CRNN [Pui17]	✓	-	9,6M	**4,4**	-	-	2,5	-	-
	CRNN [Dut+18]	-	✓	-	5,7	-	-	5,07	-	-
seq2seq	CRNN + LSTM [Mic+19]	-	-	-	5,24	-	-	-	-	-
FCN	VAN (line level) [CCP22]	-	-	2,7M	4,97	-	-	3,08	-	-
	FCN [YHM20]	-	-	3,4M	**4,9**	-	-	-	-	-
Conv + encodeur Transformer	CNN-SAN [dAr+22]	-	-	1,7M	10,26	7,50	-	-	-	-
	S-Attn + CTC [Dia+21]	-	-	12M	-	-	3,53	-	-	2,48
	S-Attn + CTC [Dia+21]	✓	-	12M	-	-	**2,75**	-	-	1,99
Conv + Transformer	CRNN-Transformer [WZG22]	-	-	4,8M	5,61	**4,15**	-	3,88[a]	3,49[a]	-
	Forward Transformer [WZG21]	-	-	13M	6,02	-	-	-	-	-
	Transformer [Dia+21]	-	-	21M	-	-	**2,96**	-	-	**2,01**
	CRNN-Transformer [WZG22]	✓	-	24M	14,38	6,46	-	-	3,39[a]	-
	Bidi. Transformer [WZG21]	-	-	27M	5,67	-	-	-	-	-
	FPHR Transformer [SK21]	-	-	28M	-	6,5	-	-	-	-
	Transformer [Kan+22]	-	-	100M	7,62	4,67	-	-	-	-
ViT + décodeur Transformer	$\text{TrOCR}_{\text{LARGE}}$ [Li+23]	✓[b]	-	558M	-	-	2,89[b]	-	-	-
Conv + encodeur Transformer	Notre CFTNN	-	✓	3,5M	6,11	5,53	-	3,45	3,18	-
Conv + Transformer	Notre VLT	-	✓	5,6M	5,67	4,23	-	3,64	2,82	-
Conv + Transformer	Notre LT	-	✓	7,7M	5,61	4,33	-	3,49	**2,51**	-

[a] Pour le jeu de données RIMES, les résultats de l'architecture CRNN-Transformer sont ici donnés avec un algorithme de décodage "CTC-Prefix-Score" en l'absence de résultat sans l'utilisation de ce dernier

[b] L'architecture TrOCR utilise un Transformer pour la vision [Bao+21] pré-entraîné sur ImageNet en tant qu'encodeur, ainsi qu'un modèle de langue large [Liu+19] pré-entraîné sur de nombreuses données en tant que décodeur. Bien que son adaptation à la reconnaissance d'écriture utilise uniquement les données annotées d'IAM ainsi que des données synthétiques, nous considérons qu'elle utilise des données annotées externes.

6,11% sur IAM sans données synthétiques et de 5,53% avec. Les architectures VLT et LT obtiennent de bons scores tout en étant relativement plus légères que les autres architectures du même type. Seule l'architecture "CRNN-Transformer" proposée par Wick et al. [WZG22] utilise moins de paramètres, notamment grâce à l'utilisation de couches récurrentes au lieu de couche d'encodeur Transformer.

Sans utiliser de données synthétiques, l'architecture VLT obtient un CER de 5,67% sur IAM et de 3,64% sur RIMES. L'architecture LT parvient quant à elle à un meilleur résultat avec un CER de 5,61% sur IAM et de 3,50% sur RIMES. Les résultats obtenus sont meilleurs que les autres architectures de Transformer, ce qui montre en particulier que nos architectures légères sont adaptées pour s'entraîner avec peu de données annotées contrairement aux architectures larges. Néanmoins, des architectures plus légères et plus simples telles que des architectures récurrentes ou FCN, parviennent à des taux d'erreur plus faibles.

Nos architectures bénéficient tout de même de l'utilisation de données synthétiques pour s'améliorer significativement. L'architecture VLT obtient ainsi d'excellents CER de 4,23% sur IAM ainsi que de 2,82% sur RIMES.

Sur RIMES, l'architecture LT obtient un CER de 2,51%, encore meilleur que celui obtenu par VLT. Les résultats montrent qu'elles sont ainsi capables de surpasser la majorité des approches Transformers, ainsi que les plus larges. Cela indique la encore qu'une architecture légère peut être suffisante et qu'une architecture large n'est pas forcément nécessaire, malgré l'utilisation de données synthétiques.

Cependant, nos approches, entraînées avec des données synthétiques, ne rivalisent pas avec des approches utilisant des données annotées réelles externes. Cela montre l'importance des données annotées réelles pour obtenir de très bons résultats.

Enfin, nous pouvons noter que les architectures VLT et LT obtiennent de bons résultats, proches de ceux obtenus par l'approche de Puigcerver [Pui17] qui utilise un modèle de langue. Cela semble indiquer que nos architectures sont capables d'apprendre à modéliser efficacement la langue, ou au moins d'obtenir des résultats similaires. Ces architectures, qui utilisent un décodeur Transformer, ont certes besoin de données synthétiques pour obtenir des résultats au même niveau, mais peuvent en revanche être entraînées de bout en bout. Elles ne nécessitent ainsi pas de modèle de langue extérieur entraîné (ou pré-entraîné dans le cas de l'architectures de Li et al. [Li+23]) sur différentes données.

10.8 Évaluation des prédictions des modèles

Dans cette section, nous présentons brièvement diverses statistiques sur les erreurs commises par nos modèles.

Pour cela, nous prenons comme référence les meilleurs modèles que nous avons entraînés sur chaque jeu de données. Pour le jeu de données IAM, l'architecture VLT entraînée avec des données synthétiques est utilisée, tandis que nous utilisons l'architecture LT, elle aussi entraînée avec des données synthétiques, pour RIMES.

10.8.1 Répartition des erreurs

Dans la table 10.16, nous présentons la répartition des CER obtenus en test.

TABLE 10.16 – Répartition des CER sur les jeux de données IAM et RIMES. Nous rappelons qu'un taux d'erreur inférieur à 10% est considéré comme acceptable pour des corrections manuelles [Str+18].

Jeux de données	CER moyen (%)	Répartition du CER			
		0%	0–5%	5–10%	>10%
IAM	4,23	39,45	28,92	18,25	13,38
RIMES	2,51	70,31	19,15	6,81	3,73

Sur le jeu de données IAM, 39,45% des prédictions ne contiennent aucune erreur, tandis que 86,62% des prédictions contiennent moins de 10% de caractères erronés. Nous fournissons un histogramme plus détaillé de la répartissons des erreurs sur ce jeu de données dans la figure 10.7.

Sur le jeu de données RIMES, qui est plus facile en comparaison, le modèle LT parvient à prédire 70,31% des écritures sans aucune faute, et seulement 3,73% des prédictions ont plus de 10% de caractères erronés. Cependant, une grande partie de ces erreurs importantes est due à des erreurs dans la vérité terrain (comme dans la figure 10.9d).

10.8.2 Analyse qualitative des prédictions

Nous présentons maintenant brièvement quelques prédictions générées par nos modèles. Dans la figure 10.8, nous comparons nos prédictions avec la vérité terrain du jeu de données IAM afin d'illustrer différents taux d'erreurs. La figure 10.9 présente des résultats similaires sur le jeu de données RIMES.

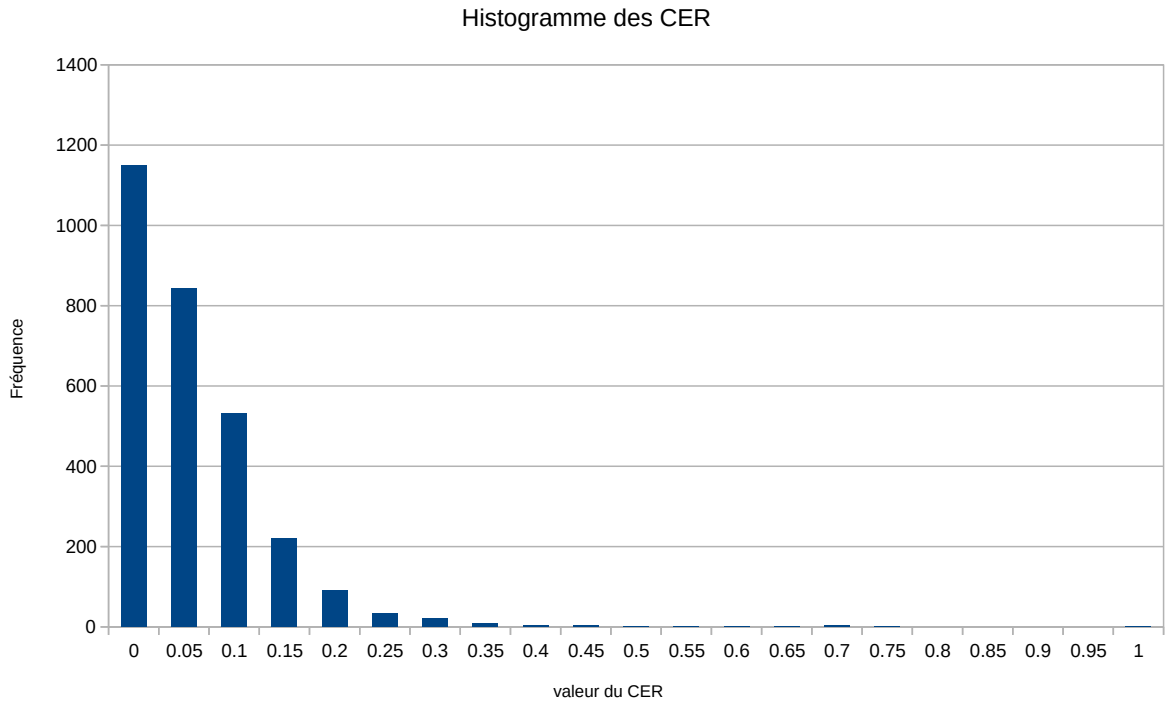
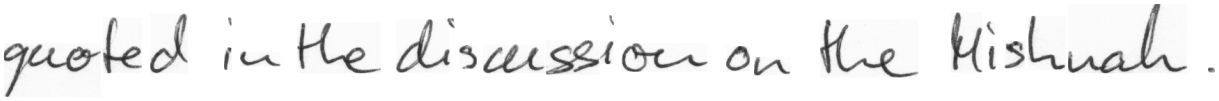


FIGURE 10.7 – Histogramme des CER obtenus par l'approche VLT entraînée avec des données synthétiques, sur le jeu de données IAM.

10.9 Synthèse des résultats

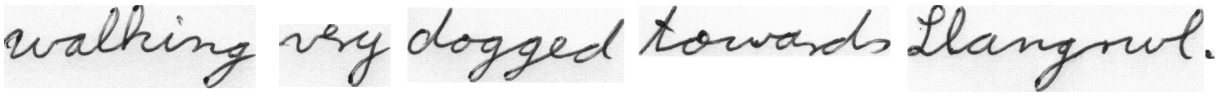
Dans cette section, nous avons évalué les modèles que nous avons proposés, et les taux d'erreur obtenus ont permis de valider leurs performances sur de l'écriture moderne. Dans un premier temps, nous avons vu que l'architecture CFTNN est capable d'obtenir des résultats globalement au niveau des approches CRNN. Nous avons de plus démontré qu'elle est particulièrement adaptée en tant qu'encodeur dans des architectures de Transformer autorégressives. Nous avons ensuite montré que les architectures de Transformer légères VLT et LT sont capables d'obtenir de bons résultats grâce à l'ajout d'un décodeur Transformer.

Pour entraîner ces architectures, les différentes stratégies proposées se sont révélées particulièrement pertinentes. Nous avons évalué l'impact des différentes stratégies. Il en est ressorti que les augmentations de données et génération de données synthétiques sont cruciales, tout comme l'utilisation d'une fonction de coût hybride. De plus, l'utilisation d'un encodage positionnel supplémentaire, ainsi que d'une stratégie de prédiction basée sur de l'augmentation en test, permettent de réduire les taux d'erreur encore plus loin.



Vérité terrain : `quoted in the discussion on the Mishnah`
Prédiction : `quoted in the discussion on the Mishnah`

(a) Exemple de ligne correctement reconnue par notre modèle. Identifiant de la ligne de texte : `d07-093-03`.

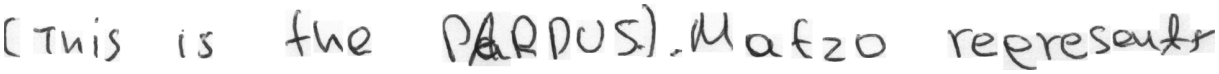


Vérité terrain : `walking very dogged towards Llangrwl.`
Prédiction : `walking very dogged towards Llangrowl.`

(b) Exemple de prédiction comportant des erreurs. Sur cet exemple, le modèle VLT obtient un CER de 2,70% et un WER de 20,00%. Identifiant de la ligne de texte : `m04-113-04`.

Vérité terrain : `On the back seat Stan Hayball embraced`
Prédiction : `On the back neat Stan Hapball embraced`

(c) Exemple de prédiction comportant des erreurs. Sur cet exemple, le modèle VLT obtient un CER de 5,26% et un WER de 28,57%. Identifiant de la ligne de texte : `m04-131-03.png`.



Vérité terrain : `(This is the PARDUS.) Matzo represents`
Prédiction : `(This is the PAROUS). Matzo represent[s]`

(d) Exemple (rare) de lignes avec de forts taux d'erreur. Par ailleurs, cet exemple comporte une légère erreur dans la vérité terrain. Sur cet exemple, le modèle VLT obtient un CER de 10,53% et un WER de 33,33%. Identifiant de la ligne de texte : `d04-111-03`.

FIGURE 10.8 – Exemples de prédictions, obtenues avec l'architecture VLT entraînée avec des données synthétiques, sur le jeu de données IAM. L'utilisation de crochets "[...]" dénote des caractères manquants dans la prédiction. Les images ont été choisies pour illustrer différents taux d'erreur.

Vérité terrain : `phonique, je vous confirme ma demande par écrit`
Prédiction : `phonique, je vous confirme ma demande par écrit`

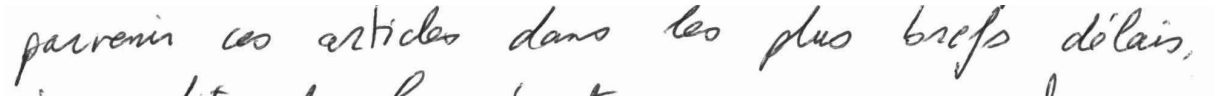
(a) Exemple de ligne correctement reconnue par notre modèle. Identifiant de la ligne de texte : `eval2011-17_0_1`.

Vérité terrain : `commander 1 mais 2 paires de chaussettes réf. 1919AB.`
Prédiction : `commander 1 mais 2 paires de chaussettes réf: 1919A8.`

(b) Exemple de prédiction comportant des erreurs. Sur cet exemple, le modèle LT obtient un CER de 3,77% et un WER de 22,22%. Identifiant de la ligne de texte : `eval2011-30_0_3`.

Vérité terrain : `JE ME PERMETS DE VOUS ECRIRE POUR AVOIR`
Prédiction : `JE ME PERMETS DE VOUS ECRIRE POUR POUVOIR`

(c) Exemple de prédiction comportant des erreurs. Sur cet exemple, le modèle LT obtient un CER de 7,69% et un WER de 12,50%. Identifiant de la ligne de texte : `eval2011-71_0_0`.



Vérité terrain : `Comptant sur votre sérieux pour me faire`
Prédiction : `parvenir ces articles dans les_plus_ brefs_délais.`

(d) Exemple de vérité terrain erronée. Dans l'état actuel de la vérité terrain, le modèle LT obtient un CER de 90,00% et un WER de 114,29%. Nous avons volontairement omis les insertions de caractères dans cet exemple. Identifiant de la ligne de texte : `eval2011-53_0_5`.

FIGURE 10.9 – Exemples de prédictions sur des images du jeu de test de RIMES. Les prédictions sont obtenues avec l'architecture LT entraînée avec des données synthétiques.

Comparées à l'état de l'art, nos architectures légères sont compétitives. Sans données synthétiques, elles obtiennent des taux d'erreur relativement corrects. Elles obtiennent des résultats bien meilleurs que les autres approches de Transformer, et restent au niveau de l'état de l'art des architectures plus simples à entraîner. Combinées avec des données synthétiques, elles parviennent à être compétitives, et obtiennent des résultats comparables aux architectures de Transformer plus larges, voire meilleurs dans certains cas. De plus, nous avons montré qu'une architecture large n'est pas forcément nécessaire et qu'une architecture légère est suffisante pour obtenir des résultats de niveau comparable, voire meilleurs. Elles restent donc bien plus simples à entraîner que des architectures de Transformer plus larges, tout en étant capables de s'entraîner avec très peu de données annotées.

Compte tenu de ces résultats, les architectures LT et VLT semblent ainsi tout à fait capables d'obtenir de bons résultats avec des données limitées et plus complexes, comme c'est le cas pour des documents anciens.

Chapitre 11

Résultats sur des documents anciens

Dans ce chapitre, nous évaluons maintenant nos architectures sur de l'écriture manuscrite ancienne. Tandis que le chapitre précédent a pour but de valider les performances de nos architectures sur des documents modernes, ici, nous montrons que nos architectures sont capables d'obtenir de bons résultats sur des documents anciens bien plus complexes. Comme expliqué dans le chapitre 5, les documents anciens possèdent en particulier une écriture généralement plus difficile à reconnaître, une langue complexe à modéliser, ainsi qu'un nombre de données annotées généralement restreint. De plus, nous démontrerons l'intérêt des données synthétiques que nous avons proposées spécifiquement pour de l'écriture manuscrite ancienne.

Pour évaluer les performances de nos modèles et stratégies sur des documents anciens, nous utilisons deux jeux de données anciens : READ 2016 et READ 2018. Ils permettent en particulier de s'évaluer sur deux scénarios distincts : l'entraînement sur des données homogènes disponibles en quantité (environ 10 000 images de lignes de texte) pour READ 2016 ; ainsi que la spécialisation de modèles sur des données spécifiques disponibles en faible quantité pour READ 2018. En conséquence, les résultats présentés dans cette section sont sujets à des variations liées à ces scénarios.

Ce chapitre commencera par présenter dans la section 11.1 le protocole expérimental utilisé pour l'évaluation des modèles. Dans la section 11.2, nous évaluerons ensuite l'apport des données synthétiques pour l'entraînement sur des données anciennes. La section 11.3 discutera quant à elle de l'intérêt d'utiliser des architectures légères. La stratégie d'entraînement ainsi que celle de prédiction, toutes deux introduites dans le chapitre 9, sont ensuite étudiées dans les sections 11.4 et 11.5 respectivement. Nous comparerons ensuite les résultats obtenus par les architectures proposées avec le reste de l'état de l'art dans la section 11.6. Nous terminerons ce chapitre par une analyse plus poussée des résultats obtenus par nos meilleurs approches dans la section 11.7.

11.1 Protocole expérimental

Dans cette section, nous présentons le protocole expérimental pour entraîner nos modèles à la tâche de reconnaissance d'écriture manuscrite et ancienne. Il est en majorité similaire au protocole utilisé dans le chapitre précédent pour des documents modernes. Nous évoquerons donc en particulier les différences avec ce dernier.

11.1.1 Jeux de données anciens

READ 2016

Le jeu de données READ 2016 est un jeu de données d'écriture manuscrite ancienne composé de documents homogènes. Il a été présenté dans la section 5.1.2 (p. 74). Il permet d'évaluer les performances de nos modèles entraînés sur ces données homogènes et en quantité plus importante, comparé aux documents anciens classiques.

READ 2018

Le jeu de données READ 2018 permet quant à lui d'évaluer les performances de nos architectures sur un scénario où des données génériques sont disponibles en quantité suffisante (environ 12 000 images de lignes de texte) pour entraîner un premier modèle, qui servira ensuite de base pour un réentraînement sur des jeux de données spécifiques et en quantité limitée. Il est particulièrement intéressant puisqu'il propose une évaluation sur 5 jeux de données spécifiques. Plus de détails sont disponibles dans la section 5.1.2 (p. 76).

Nos modèles sont tout d'abord entraînés sur les données génériques de READ 2018. En revanche, nous appliquons diverses corrections sur les données génériques de READ 2018 :

— nous supprimons les données sur lesquelles la segmentation en lignes ne fait pas de sens, résultant en l'absence de lignes de textes claires ;

— et nous corrigeons les erreurs d'appariement des vérités terrains avec les images de lignes correspondantes.

Nous utilisons la répartition suivante sur ces données corrigées en répartissant 10% des données choisies aléatoirement pour l'ensemble de validation : 10 712 images de lignes de texte pour entraîner nos modèles ; ainsi que 1 172 images en validation [1].

Nos modèles sont ensuite adaptés sur chacun des 5 jeux de données spécifiques, en utilisant soit aucune (0), 1, 4 ou 16 pages de l'ensemble des lignes de textes annotées du document spécifique. Les documents spécifiques étant composés de relativement peu de données annotées (entre 20 et 70 lignes avec 1 page, et entre 300 et 1000 lignes avec 16 pages annotées), nous utilisons une validation croisée à 4 blocs pour entraîner nos modèles [2].

Dans les résultats présentés sur le jeu de données READ 2018, nous donnons la moyenne des taux d'erreur obtenus sur les différents blocs de validation croisée. De plus, nous fournissons un écart-type mesuré sur les 4 blocs de validation croisée. Notons que parler d'écart-type ne fait pas forcément sens comme les données d'entraînement et de validation diffèrent selon les blocs. Cependant, nous pensons que ce pseudo-écart-type reste quand même une donnée intéressante pour évaluer la sensibilité du modèle.

11.1.2 Données synthétiques

Pour entraîner les architectures proposées, nous utilisons des données synthétiques. Dans ce chapitre, les résultats sont soit présentés avec des données synthétiques globalement génériques, soit des données synthétiques paramétrées au style visuel du jeu de données READ 2016.

Dans le but d'améliorer la capacité à apprendre des contenus à la fois graphiques et textuels variés, nous proposons dans un premier temps des données synthétiques génériques. Le processus pour générer ces données a été décrit dans la section 8.2 (p. 128) de ce manuscrit. Nous utilisons donc :

— 21 polices d'écritures manuscrites au style des documents anciens ;

— du texte issu de Wikipédia, de livres disponibles en ligne, ainsi que la vérité terrain du jeu de données considéré ;

— ainsi que diverses augmentations et dégradations.

1. La répartition proposée, ainsi que la vérité terrain corrigée avec la liste des modifications, sont disponibles à l'adresse suivante : `https://gitlab.inria.fr/intuidoc-public/vlt`

2. De même, ce découpage est fourni à l'adresse suivante : `https://gitlab.inria.fr/intuidoc-public/vlt`

Ces données synthétiques sont utilisées pour entraîner des modèles sur le jeu de données READ 2016, ainsi que lors de la phase d'entraînement sur les données génériques de READ 2018. En revanche, nous rappelons que nous n'utilisons pas de données synthétiques lors de la phase de spécialisation d'un modèle sur des données spécifiques de READ 2018. En effet, nous visons à adapter nos modèles à l'écriture de l'auteur du document spécifique.

11.1.3 Procédure d'entraînement et de prédiction

Entraînement

Stratégie d'entraînement Pour le jeu de données READ 2018, nous appliquons la stratégie d'entraînement proposée dans la section 9.1 (p. 139). Nous rappelons que celle-ci propose en particulier un entraînement en trois parties :

1. un entraînement sur les données génériques en utilisant les ensembles d'entraînement et de validation séparés ;
2. un entraînement sur les données génériques sans validation, en intégrant l'ensemble de validation dans les données d'entraînement ;
3. une adaptation sur les données spécifiques de l'encodeur, tout en figeant les poids de la partie décodeur.

Bien qu'il pourrait être intéressant d'utiliser plus de données annotées (provenant par exemple d'autres jeux de données annotées), nous choisissons de limiter les données réelles aux données fournies pour respecter le cadre de la compétition, et ainsi permettre une comparaison équitable. Nous utiliserons en revanche des données synthétiques qui sont très peu coûteuses en comparaison à des données annotées. Cette stratégie d'entraînement n'est en revanche pas appliquée sur le jeu de données READ 2016 qui n'offre pas un scénario adéquat.

Conditions d'arrêt Pour le jeu de données READ 2016, l'entraînement suit la même procédure que celle décrite dans la section 10.1.4 du chapitre précédent (p. 156). Sur le jeu de données READ 2018, l'entraînement est arrêté après 200 époques sans amélioration du CER lors de la première phase d'entraînement. Le modèle ayant obtenu le CER le plus faible est ensuite entraîné durant 10 époques consécutives sans ensemble de validation lors de la deuxième phase d'entraînement sur les données génériques. Lors de la phase d'adaptation sur des données spécifiques, nos modèles sont entraînés jusqu'à observer 50 époques sans amélioration du CER.

Stratégie de prédiction

La procédure d'évaluation est similaire à celle présentée dans la section 10.1.4 (p. 156). De plus, nous utilisons la stratégie de prédiction présentée dans la section 9.2 (p. 143) pour évaluer les performances finales de nos modèles sur les jeux de données READ 2016 et READ 2018. Seuls les résultats présentés dans les sections 11.5 et 11.6 feront donc usage de cette stratégie.

11.2 Intérêt des données synthétiques

Dans cette section, nous montrons que nos données synthétiques sont importantes pour entraîner des architectures de Transformer sur la reconnaissance d'écriture manuscrite ancienne. Les tables 11.1 et 11.2 résument les résultats obtenus pour les jeux de données READ 2016 et READ 2018 respectivement, en entraînant nos architectures avec ou sans des données synthétiques.

Table 11.1 – Impact de l'utilisation de données synthétiques sur les taux de reconnaissance pour des modèles entraînés sur le jeu de données READ 2016. Ces données synthétiques (génériques) sont identiques à celles proposées dans la section 11.1.2.

Architecture	READ 2016		READ 2016 + synth. (génériques)	
	CER (%)	WER (%)	CER (%)	WER (%)
CRNN	6,08	27,63	5,96	28,37
CFTNN	6,07	28,45	6,01	29,36
VLT	6,88	28,73	5,68	24,98
LT	6,17	26,60	**5,49**	**24,68**

Les différents résultats indiquent une réduction notable des taux d'erreur lorsque nous utilisons des données synthétiques pour entraîner nos architectures. En effet, nous observons une réduction du CER apportée par l'utilisation de données synthétiques allant jusqu'à 18%. En particulier, nous observons de plus forts gains sur les architectures VLT et LT, qui utilisent toutes deux un décodeur Transformer. Ce résultat rejoint les conclusions obtenues sur des documents modernes.

Les données synthétiques nous semblent en particulier cruciales pour entraîner efficacement des architectures Transformer basées sur l'utilisation d'un décodeur. En effet, sans données synthétiques, nous avons notamment observé des performances du décodeur en dessous de celles offertes par le décodeur de l'architecture. Cela indique que la quantité de

TABLE 11.2 – Impact de l'utilisation de données synthétiques sur les taux de reconnaissance pour des modèles entraînés sur le jeu de données READ 2018. Les taux d'erreur présentés correspondent aux moyennes sur l'ensemble des 5 documents spécifiques et des différentes quantités de données annotées disponibles.

Architecture	READ 2018		READ 2018 + synth.	
	CER (%)	WER (%)	CER (%)	WER (%)
CRNN	16,21	49,21	14,84	47,01
CFTNN	17,73	52,97	15,27	49,51
VLT	17,73	48,21	**13,67**	**41,52**
LT	17,15	46,39	14,29	42,23

données annotées disponible initialement sur des jeux de données anciens reste insuffisante pour entraîner des décodeurs Transformer.

Les données synthétiques proposées semblent ainsi adaptées pour entraîner efficacement des architectures de Transformer sur des écritures manuscrites anciennes.

Pour aller plus loin, nous proposons diverses expériences. Dans la section 11.2.1 nous montrerons l'intérêt des différents types de contenu textuel utilisés. Nous étudierons ensuite l'impact du style visuel obtenu à partir d'augmentations et de dégradations dans la section 11.2.2. Enfin, dans la section 11.2.3, nous étudierons l'impact d'utiliser des données synthétiques génériques ou paramétrées au jeu de données READ 2016.

11.2.1 Quel contenu textuel pour la génération de données synthétiques

Les données synthétiques que nous proposons pour des écritures anciennes sont composées de différents types de contenus textuels. En plus de contenus textuels anciens, nous utilisons aussi des articles Wikipédia dans le but d'enrichir ce contenu textuel. Le vocabulaire utilisé, ainsi que la langue ayant subi des évolutions notables, nous proposons d'étudier l'apport de chaque type de contenu textuel.

Pour évaluer cet impact, nous comparons l'architecture VLT entraînée sur les données génériques de READ 2018, ainsi que des données synthétiques générées à partir du contenu textuel provenant :

— de la vérité terrain de READ 2018 (ensemble d'apprentissage des données génériques ;

— d'articles Wikipédia ;

— d'e-books anciens ;

— ou de l'ensemble des trois contenus textuels précédents.

Chaque modèle est ensuite adapté sur l'ensemble des documents spécifiques et des données annotées disponibles. Les résultats sont ensuite évalués sur l'ensemble de test des données spécifiques, et nous proposons dans la table 11.3, la moyenne des taux d'erreur obtenus sur les scénarios proposés (0–16 pages).

TABLE 11.3 – taux d'erreur obtenus par l'architecture VLT sur le jeu de données READ 2018 lors d'un pré-entraînement sur les données génériques ainsi que des données synthétiques avec différents contenus textuels. Les résultats sont donnés sous la forme d'une moyenne sur l'ensemble des scénarios (0–16 pages), conformément à la compétition sur le jeu de données READ 2018.

Contenu textuel utilisé pour des données synthétiques	CER (%) moyen	WER (%) moyen
Pas de données synthétiques	17,73 ± 1,30	48,21 ± 2,38
Vérité terrain READ 2018	14,26 ± 0,55	**40,92** ± 2,61
Articles Wikipedia	15,70 ± 1,37	44,67 ± 2,49
E-books anciens	15,60 ± 1,16	44,07 ± 2,58
Tout les contenus	**13,67** ± 0,50	41,52 ± 1,69

Nous comparons pour l'instant l'impact des trois contenus textuels proposés. L'utilisation de vérité terrain de READ 2018 fournit les taux d'erreur les plus faibles en comparaison aux autres approches. L'utilisation d'articles Wikipédia ou encore d'e-books anciens permet d'enrichir le contenu textuel vu par les modèles. Néanmoins, les modèles entraînés avec ces contenus textuels obtiennent des taux d'erreur légèrement inférieurs à ceux obtenus en utilisant la vérité terrain. Cela peut s'expliquer par la nature des contenus textuels qui diffère des données spécifiques. Néanmoins, il est à noter que l'utilisation de ces contenus textuels reste une largement meilleure solution qu'un entraînement sans données synthétiques.

Les résultats obtenus indiquent qu'il n'est pas forcément nécessaire d'utiliser des contenus extérieurs pour obtenir de bons résultats. Le simple fait d'utiliser la vérité terrain du jeu de données montre une amélioration relative du CER de 20%. Celle-ci est en particulier due à la combinaison des contenus textuels provenant de lignes consécutives, ce qui permet d'apprendre de nouveaux contextes.

Enfin, nous observons un meilleur CER en fusionnant l'ensemble des contenus textuels, ce qui permet à la fois de fournir un contenu textuel enrichi, mais aussi plus proche des documents spécifiques. Nous pouvons aussi noter, que l'utilisation de données synthétiques permet de réduire la variance dans les résultats observés. En revanche, il est à noter que le WER est moins bon que lorsque nous utilisons uniquement la vérité terrain de READ 2018. Nous pensons que cela peut être dû, à l'apprentissage de mots modernes, qui viendrait dégrader ce taux de reconnaissance sur des documents spécifiques anciens.

11.2.2 Impact des transformations visuelles

Nous évaluons maintenant l'impact du style visuel des données synthétiques sur les taux d'erreur. Pour cela, nous comparons les résultats obtenus avec notre architecture VLT, entraînée sur les données génériques et des données synthétiques augmentées ou non. Nous proposons de comparer trois cas :

— Avec des **polices** d'écritures **seules**, où des données synthétiques sont générées, mais aucune augmentation ou dégradation n'est utilisée. Un exemple de telles données synthétique est disponible dans la figure 8.4 (2.) (p. 130).

— Avec l'utilisation d'**augmentations**, où uniquement des augmentations (dilatation, érosion, distorsion élastique, italique, rotation, transformation de la perspective, changements de la luminosité, du contraste et de la netteté) sont utilisées pour déformer les polices d'écritures. Ce type de données synthétiques est illustré dans la figure 8.4 (3.) (p. 130)

— En utilisant des **augmentations et dégradations** combinées, qui ajoutent en plus diverses dégradations de l'image pour donner un style ancien réaliste (taches d'encre, coloration de l'encre, effacement de l'encre, ajout d'un fond aléatoire au document, et ajout d'un bruit gaussien). La figure 8.4 (4.) (p. 130) montre un exemple d'image synthétique.

Impact des transformations visuelles sur READ 2016

Nous proposons dans un premier temps d'évaluer l'impact des augmentations et dégradations sur le jeu de données READ 2016, qui offre un cadre d'entraînement sans spécialisation. La table 11.4 présente les résultats obtenus avec les trois cas.

Les augmentations de données permettent une amélioration nette du CER de 4% relativement. L'utilisation de dégradations en plus d'augmentations apporte quant à elle

TABLE 11.4 – Impact de l'utilisation d'augmentations et de dégradations influençant le style visuel des données synthétiques sur le jeu de données READ 2016.

Types de données synthétiques	READ 2016	
	CER (%)	WER (%)
Polices seules	5,70	25,97
Augmentations	5,47	25,14
Augmentations et dégradations	5,15	24,68

une réduction supplémentaire du CER de 6% relativement. L'utilisation de données synthétiques augmentées et dégradées est donc particulièrement pertinente pour le jeu de données READ 2016 qui contient des dégradations comme par exemple de l'effacement de l'encre. De plus, cela indique que notre approche de génération de données synthétiques pour des document anciens est bien adaptée.

Impact des transformations visuelles sur READ 2018

Nous étudions ensuite les résultats sur le jeu de données READ 2018, qui propose de voir l'impact des données synthétiques dans un scénario de spécialisation. Les différents types de données synthétiques sont ainsi testés lors de la phase d'entraînement sur des données génériques. Les résultats sont quant à eux évalués sur l'ensemble de test des données spécifiques, après une phase d'adaptation sur des pages annotées, si disponibles. La table 11.5 présente ainsi les résultats obtenus.

TABLE 11.5 – Impact de l'utilisation d'augmentations et de dégradations influençant le style visuel des données synthétiques sur le jeu de données READ 2018.

Types de données synthétiques	CER (%) moyen par pages annotées				CER (%) moyen (0–16 pages) par document					CER (%) moyen
	0 page	1 page	4 pages	16 pages	Konzil. C	Schiller	Ricordi	Patzig	Schwerin	
Polices seules	27,17	15,08	9,86	6,47	6,89	15,37	18,56	19,39	12,12	14,65 ± 0,75
Augmentations	26,91	**13,45**	**9,23**	**6,06**	6,79	15,12	17,31	17,94	**11,74**	13,91 ± 0,57
Augmentations et dégradations	**25,17**	13,78	9,52	6,19	**6,48**	**14,92**	**16,14**	**17,49**	12,24	**13,67** ± 0,50

En comparaison avec des données synthétiques obtenues uniquement à partir de polices d'écritures seules, les augmentations de données montrent une amélioration claire des résultats. Cette amélioration s'observe dans tous les cas, peu importe le document de

spécialisation considéré, ou le nombre de pages annotées disponible pour se spécialiser. En moyenne sur l'ensemble des documents et des nombres de pages annotées, l'utilisation d'augmentations permet ainsi de passer d'un CER de 14,65 à 13,91, soit une amélioration relative de 4%.

En ajoutant des dégradations en plus de l'augmentation de données sur les données synthétiques, nous observons une réduction du taux d'erreur moyen de 13,91 à 13,67 (soit une amélioration de 2% relativement). En particulier, ce gain dépend du document considéré. Ainsi, sur le document spécifique Ricordi, nous observons par exemple un gain important grâce à l'utilisation de données synthétiques dégradées lors de l'entraînement. En effet, les données de test de ce document sont particulièrement dégradées, ce qui rend nos données synthétiques dégradées pertinentes. Nous notons en particulier que l'amélioration est d'autant plus prononcée avec aucune page pour s'adapter (réduisant le CER de 30,38 à 28,11, soit une amélioration relative de 7%). Cependant, pour le document spécifique Schwerin, nous observons l'effet inverse, avec des taux d'erreur moins bons (4% relativement). Cette différence peut s'expliquer par le fait que les données du document spécifiques contiennent très peu de dégradations ainsi qu'une encre globalement bien préservée.

Enfin, nous notons une amélioration particulièrement notable lorsqu'aucune page annotée (0 page) n'est disponible pour spécialiser les modèles. L'utilisation de données synthétiques dégradées apporte 6% relativement.

Avec des pages annotées, nous notons en revanche que l'utilisation de dégradations se révèle être inefficace et conduisant à l'inverse à une dégradation des taux d'erreur. Cet impact, bien que fluctuant selon le document spécifique considéré, peut notamment s'expliquer par le fait que la partie optique de notre architecture est réentraînée, et donc spécialisée sur un type de document. Des dégradations peuvent ainsi apporter d'importantes variations, qui ne sont pas forcément bénéfiques pour la spécialisation de tous les documents.

L'utilisation de dégradations pour les données synthétiques apportant tout de même une réduction du CER moyen ainsi que du CER sur les scénarios les plus complexes (c'est-à-dire sans pages annotées pour se spécialiser ou sur le document spécifique Ricordi), nous choisissons de conserver l'utilisation des dégradations.

11.2.3 Impact de la variabilité dans les données synthétiques

Données synthétiques paramétrées à READ 2016

Le jeu de données READ 2016 est constitué de données relativement homogènes. Notre algorithme de génération de données synthétiques étant paramétrable, nous proposons ainsi la génération de données synthétiques paramétrées pour ressembler au style des documents de READ 2016.

Pour ce faire, nous utilisons 16 polices d'écritures [3] au style des écritures manuscrites anciennes, que nous avons sélectionnées manuellement, car nous les jugeons suffisamment proche des écritures du jeu de données. Nous utilisons les mêmes augmentations et dégradations que celles appliquées pour la génération de données synthétiques génériques avec quelques modifications. l'inclinaison des caractères (écriture italique) est moins prononcée (rotation aléatoire comprise entre 0° et 20°), et nous utilisons uniquement de la dilatation de l'écriture pour grossir les tracés (l'érosion de l'écriture n'est pas utilisée). Nous utilisons des fonds directement extraits des images de pages du jeu de données. Le contenu textuel utilisé reste quant à lui inchangé.

Des exemples de ces données synthétiques sont illustrés dans la figure 11.1a, et peuvent être comparées aux données réelles de la figure 11.1b.

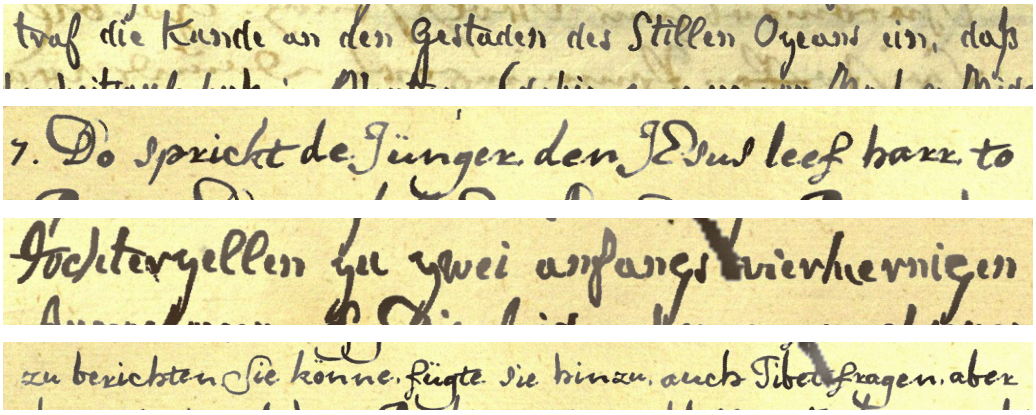
Dans la table 11.6, nous comparons les taux d'erreur selon le type de données synthétiques utilisées à l'entraînement. Notons que le contenu textuel reste quant à lui similaire.

Table 11.6 – Impact de l'utilisation de différents types de données synthétiques pour le jeu de données READ 2016.

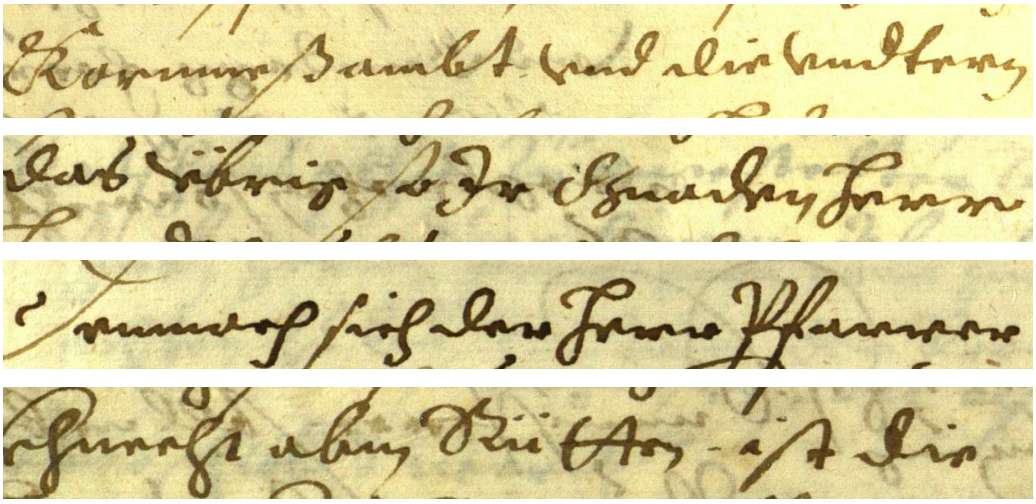
Architecture	READ 2016		READ 2016 + synth. génériques		READ 2016 + synth. paramétrées	
	CER (%)	WER (%)	CER (%)	WER (%)	CER (%)	WER (%)
CRNN	6,08	27,63	5,96	28,37	5,65	26,99
CFTNN	6,07	28,45	6,01	29,36	6,01	29,03
VLT	6,88	28,73	5,68	24,98	**5,15**	24,68
LT	6,17	26,60	5,49	24,68	5,31	**24,59**

Les résultats indiquent que les données synthétiques paramétrées pour ressembler au style visuel de READ 2016 offrent de meilleurs résultats sur les différentes architectures proposées. Néanmoins, nous notons tout de même des gains importants avec des données synthétiques génériques par rapport à un entraînement sans données synthétiques. De

3. Ces polices d'écritures sont disponible à l'adresse suivante : `https://gitlab.inria.fr/intuidoc-public/synthetic-handwriting-generation`



(a) Images de lignes de texte synthétiques manuscrites au style du jeu de données READ 2016.



(b) Images de lignes de texte synthétiques manuscrites au style du jeu de données READ 2016.

FIGURE 11.1 – Les images de lignes de texte synthétiques manuscrites au style du jeu de données READ 2016 que nous proposons (a), sont visuellement similaires aux lignes réelles du jeu de données READ 2016 (b)

plus, les performances obtenues en utilisant des données synthétiques génériques restent bonnes et proches de celles obtenues avec des données synthétiques paramétrées.

Nous observons que l'architecture VLT profite particulièrement de l'utilisation de données synthétiques paramétrées, et qu'elle obtient les meilleurs résultats comparée aux autres architectures testées. Selon nous, cet écart est du au fait que l'encodeur soit bien plus facile à entraîner sur un style visuel précis. Nous le verrons en particulier plus tard, dans la section 11.3.

L'architecture LT possède quant à elle plus de couches de décodeur. Elle nous semble donc plus pertinente pour modéliser un vocabulaire varié ainsi que pour agréger de l'information sur des caractéristiques variées obtenues sur différentes images. Elle obtient ainsi de meilleurs résultats que l'architecture VLT lorsque celle-ci est entraînée avec des données synthétiques génériques.

Compte tenu des résultats obtenus, dans le cas d'un jeu de données homogènes, l'utilisation de données synthétiques paramétrées semble être plus intéressant que l'utilisation de données synthétiques standard (génériques). Néanmoins, le choix de ces paramètres est réalisé manuellement, est peut être inexact. Comme nous l'avons vu, des données synthétiques standards restent tout de mêmes adaptées dans ces cas là. Nous pensons qu'elles sont en particulier utiles pour mieux généraliser.

11.2.4 Discussions sur les données synthétiques

Dans cette section, nous avons démontré l'intérêt des données synthétiques pour entraîner des architectures de réseaux de neurones, et en particulier des architectures de Transformer, sur de l'écriture manuscrite ancienne.

Nous avons vu que l'utilisation d'un contenu textuel adapté semble très important pour générer des données synthétiques utiles à de tels modèles, et en particulier les modèles utilisant un décodeur Transformer. Le style visuel permet aussi une réduction des taux d'erreur, mais avec cependant un impact globalement moins important que le contenu textuel. L'utilisation de données synthétiques augmentées permet une bonne réduction du CER, tandis que l'utilisation de dégradations dépend grandement du type de données. D'autre part, nous avons vu que dans le cadre de jeu de données homogènes, l'utilisation de données synthétiques paramétrées visant à imiter ce style visuel permet d'obtenir de meilleurs résultats que des données synthétiques standards.

11.3 Intérêt d'une architecture avec peu de poids

Dans cette section, nous discutons de l'intérêt d'utiliser des architectures légères, avec peu de poids entraînables. Comme discuté dans le chapitre 10, des architectures légères auront de nombreux avantages comme par exemple un faible temps d'entraînement ; de plus faibles temps de prédictions ; ainsi que la capacité d'être entraînées avec peu de données annotées. En particulier, les architectures légères sont particulièrement intéressantes dans le cadre des documents anciens pour leur capacité à apprendre, que ce soit pour un entraînement ou une spécialisation, avec peu de données annotées.

Table 11.7 – Résumé des CER obtenus avec nos différentes architectures sur le jeu de données READ 2018.

Architecture	CER (%) moyen par pages annotées				CER (%) moyen (0–16 pages) par document					CER (%) moyen
	0 page	1 page	4 pages	16 pages	Konzil. C	Schiller	Ricordi	Patzig	Schwerin	
CFTNN	26,48	16,40	10,83	7,37	8,38	17,36	16,06	21,13	12,21	15,27 ± 0,87
VLT	**25,17**	**13,78**	9,52	**6,19**	**6,48**	**14,92**	16,14	**17,49**	12,24	**13,67** ± 0,50
LT	26,57	14,89	**9,43**	6,29	7,19	16,03	**15,86**	19,30	**11,93**	14,29 ± 0,72

La table 11.7 propose des résultats obtenus sur READ 2018 avec les différentes architectures proposées, tandis que les résultats sur READ 2016 sont disponibles dans la table 11.1 (p. 193). Sur les deux jeux de données étudiés, l'architecture VLT montre de meilleurs résultats que l'architecture LT qui contient plus de poids. Elle s'en sort aussi bien mieux que l'architecture CFTNN qui est certes bien plus rapide, mais qui n'utilise pas de décodeur pour réduire les taux d'erreur.

Sur le jeu de données READ 2018, nous avons noté que l'architecture LT est bien plus exposée à un surapprentissage sur les données génériques. (Sur l'ensemble de validation utilisé lors de l'entraînement générique, nous observons que l'architecture LT obtient un CER de 6,53, ce qui est meilleur que le CER de 7,20 obtenu par l'architecture VLT.) L'architecture VLT montre cependant une bien meilleure capacité à se spécialiser sur des données spécifiques et en particulier avec peu de pages annotées (voir la table 11.7). Cette différence peut s'expliquer par la différence de poids dans les décodeurs, ce qui résulte ainsi en un surapprentissage de l'architecture LT sur le vocabulaire des données génériques.

L'architecture VLT combine ainsi les avantages d'un décodeur Transformer pour modéliser la langue, tout en restant légère. Elle est donc assez peu exposée à des problématiques de surapprentissage, tout en étant plus simple à entraîner. En effet, comparé à l'architecture LT, VLT possède moins de couches dans la partie décodeur. L'encodeur tire alors

parti de gradients légèrement plus pertinents en plus d'une fonction de coût hybride, et ceci malgré l'utilisation de connexions résiduelles.

11.4 Impact de la stratégie d'entraînement

Nous évaluons maintenant l'impact de la stratégie d'entraînement proposée dans la section 9.1 (p. 139). Pour cela, nous évaluons les résultats obtenus en utilisant ou non la stratégie d'entraînement sur le jeu de données READ 2018. Pour rappel, la stratégie d'entraînement propose une étape d'entraînement additionnelle sur les données génériques en utilisant l'ensemble de validation, ainsi qu'une spécialisation de l'encodeur uniquement. Nous évaluons uniquement les architectures VLT et LT comme elles utilisent à la fois un encodeur et un décodeur Transformer. La table 11.8 présente les résultats obtenus en effectuant ou non les différentes étapes de la stratégie d'entraînement présentée dans la table 9.1 (p. 141).

TABLE 11.8 – Impact de la stratégie d'entraînement proposée sur les CER moyens. Nous évaluons d'une part l'impact d'une seconde étape d'entraînement sur les données génériques incluant l'ensemble de validation dans les données d'entraînement, ainsi que le fait de figer les poids du décodeur lors de la spécialisation sur les données spécifiques.

Entraînement sur la validation générique	Décodeur figé lors de la spécialisation	Architecture	CER (%) moyen par pages annotées				CER (%) moyen
			0	1	4	16	
-	-	VLT	**25,07**	14,87	11,06	7,33	14,59 ± 2,28
		LT	26,57	16,63	11,08	7,43	15,43 ± 1,61
✓	-	VLT	**25,07**	14,84	10,57	7,26	14,43 ± 1,98
		LT	26,20	16,17	10,91	7,36	15,16 ± 0,64
✓	✓	VLT	**25,07**	**13,78**	9,52	**6,19**	**13,67** ± 0,50
		LT	26,20	14,89	**9,43**	6,29	14,29 ± 0,72

L'utilisation des données de validation lors de l'entraînement sur les données génériques semble légèrement bénéfique. Cela apporte une réduction relative du CER de 1% pour l'architecture VLT. L'architecture LT semble être un peu plus impactée, avec une réduction relative du CER de 2%, mais avec en revanche une forte réduction de la variance du CER moyen. Nous pensons que cela s'explique par un besoin plus important en données annotées pour l'architecture LT que pour l'architecture VLT.

En fixant les poids du décodeur lors de l'étape de spécialisation, nous observons une réduction du CER importante pour l'architecture VLT (5% relativement), ainsi que pour l'architecture LT (6% relativement). De plus, nous observons une réduction de la variance empirique pour l'architecture VLT ainsi que pour l'architecture LT dans une plus faible mesure.

Comme vu dans le chapitre précédent, l'entraînement d'un décodeur utilisant uniquement 2 couches est complexe. Réentraîner le décodeur de l'architecture VLT avec peu de données annotées est alors fortement sujet à du surapprentissage. Le fait de conserver les poids du décodeur entraîné sur les données génériques permet d'obtenir de meilleurs résultats, tout en diminuant fortement la variance.

Sur le jeu de données READ 2016, nous avons essayé d'appliquer une stratégie similaire se limitant à un entraînement de quelques époques avec le jeu de validation en plus du jeu d'entraînement. Cependant, nous n'avons pas observé de meilleurs taux d'erreur en procédant de la sorte. L'utilisation classique d'un ensemble de validation sur ce jeu de données semble bénéfique afin éviter un léger surapprentisage.

11.5 Impact de la stratégie de prédiction

Nous évaluons maintenant l'impact de la stratégie de prédiction appliquée à nos architectures pour la reconnaissance d'écriture manuscrite ancienne. Nous rappelons que cette stratégie consiste à utiliser de l'augmentation en test sur des images dupliquées, avant de fusionner les résultats au moyen d'un algorithme de vote. Cet impact est mesuré sur les jeux de données READ 2016 et READ 2018.

11.5.1 Impact de la stratégie de prédiction sur READ 2016

La table 11.9 indique les résultats obtenus par nos architectures sur le jeu de données READ 2016 en utilisant ou non la stratégie de prédiction.

Elle permet de réduire les CER avec une amélioration relative de 8% pour l'architecture CFTNN et de 3% pour l'architecture VLT. Pour l'architecture LT, nous n'observons pas de différence significative du CER, tandis que le WER semble être impacté négativement par la stratégie de prédiction. Nous pensons que l'architecture LT, qui utilise plus de paramètres que les autres architectures présentées, est surentraînée sur les données d'apprentissage. Elle est donc moins capable de généraliser, et obtient de moins bonnes performances sur des données de test augmentées.

TABLE 11.9 – Impact de la stratégie de prédiction que nous proposons (avec 60 prédictions) sur le jeu de données READ 2016.

Architecture	Stratégie de prédiction	READ 2016	
		CER (%)	WER (%)
CFTNN	-	6,01	29,03
	Notre (60)	**5,56**	**27,49**
VLT	-	5,15	24,68
	Notre (60)	**4,97**	**24,23**
LT	-	5,31	**24,59**
	Notre (60)	**5,30**	25,26

11.5.2 Impact de la Stratégie de prédiction sur READ 2018

TABLE 11.10 – Impact de notre stratégie de prédiction proposée (avec 60 prédictions) sur le jeu de données READ 2018.

Stratégie de prédiction	Architecture	CER (%) moyen par pages annotées				CER (%) moyen (0–16 pages) par document					CER (%) moyen
		0	1	4	16	Konzil. C	Schiller	Ricordi	Patzig	Schwerin	
CFTNN	-	26,48	16,40	10,83	7,37	8,38	17,36	16,06	21,13	12,21	15,27 ± 0,87
	Notre (60)	**25,06**	**15,62**	**10,25**	**6,78**	**7,72**	**16,44**	**15,27**	**20,19**	**11,37**	**14,43** ± 0,88
VLT	-	25,17	13,78	9,52	6,19	6,48	14,92	16,14	17,49	12,24	13,67 ± 0,50
	Notre (60)	**24,28**	**13,03**	**8,89**	**5,64**	**6,16**	**13,91**	**15,21**	**16,65**	**11,71**	**12,96** ± 0,42
LT	-	26,57	14,89	9,43	6,29	7,19	16,03	15,86	19,30	11,93	14,29 ± 0,72
	Notre (60)	**25,74**	**13,33**	**8,84**	**5,53**	**6,61**	**14,44**	**14,95**	**18,02**	**11,46**	**13,36** ± 0,45

Dans la table 11.10, nous présentons les résultats obtenus en utilisant la stratégie de prédiction sur le jeu de données READ 2018. Peu importe le scénario considéré, les architectures proposées obtiennent toutes de meilleurs résultats en utilisant la stratégie de prédiction. Elle permet une réduction relative du CER allant de 4 à 7% globalement.

Nous notons que la stratégie de prédiction est particulièrement importante pour les scénarios de spécialisation avec une réduction relative du CER allant jusqu'à 10%. Sans spécialisation (0 page), la stratégie se révèle être moins importante en comparaison, même si elle permet tout de même une amélioration du CER (3% relativement). Selon nous, ces différences d'impacts sont liées à la sensibilité des modèles aux variations dans les images lorsqu'ils sont entraînés avec peu de données. La stratégie de prédiction permet alors de lisser ces erreurs, et d'obtenir une meilleure prédiction en moyenne.

Quel que soit le jeu de données considéré, nous observons que la stratégie de prédiction se montre efficace pour réduire les taux d'erreur. Elle permet de réduire de manière importante les erreurs commises sur des documents anciens pourtant complexes. L'architecture VLT, qui a l'avantage de permettre une modélisation de la langue efficace tout en étant légère, en profite tout particulièrement, et nous obtenons les meilleurs résultats avec celle-ci.

11.6 Comparaison avec l'état de l'art

11.6.1 READ 2016

Nous nous comparons avec les approches de l'état de l'art sur le jeu de données READ 2016. Ce jeu de données propose une procédure d'entraînement plus simple en comparaison à READ 2018. Nos modèles sont entraînés sur les données d'entraînement ainsi que sur les données synthétiques paramétrées au style du jeu (proposées dans la section 11.2.3, p. 199). Les modèles sont ensuite évalués en utilisant la stratégie de prédiction proposée.

Les résultats sont présentés dans la table 11.11. En utilisant uniquement les données fournies par le jeu de données, nous obtenons des CER de 5,84%, 6,59% et 6,14% pour les architectures CFTNN, VLT et LT respectivement. Ces taux d'erreur sont globalement corrects, mais cependant inférieurs à ceux obtenus par d'autres architectures. L'architecture CRNN proposée par BYU [San+16] ou FCN proposée par Coquenet et al. [CCP22] par exemple, sont bien plus légères et simples en comparaison à nos architectures Transformer. Elles obtiennent de bien meilleurs résultats avec un CER de 5,1% et de 4,10% respectivement.

En considérant l'utilisation de données synthétiques lors de l'entraînement, nos architectures obtiennent de bien meilleurs taux d'erreur. Ces architectures étant basées sur des couches de Transformer, elles profitent particulièrement de l'ajout de données annotées supplémentaires. L'architecture VLT obtient ainsi un excellent CER de 4,97%, ce qui la place en deuxième position derrière le FCN proposé par Coquenet et al. [CCP22]. Cependant, les résultats restent à nuancer du fait que les autres architectures peuvent aussi tirer parti de l'ajout de données synthétiques, même si ce gain est certainement inférieur à celui des architectures Transformer.

Nos architectures de Transformer obtiennent de très bons résultats sur ce jeu de données. Cependant, sur ce type de données globalement homogènes, une architecture com-

TABLE 11.11 – Comparaison de nos approches avec les méthodes de l'état de l'art sur le jeu de données READ 2016. Nous indiquons les approches qui utilisent un modèle de langue (LM), ainsi qu'une stratégie de prédiction (strat. pred.).

Type d'architecture	Méthode	LM	strat. pred.	# params.	READ 2016		READ 2016 + synth.		READ 2016 + ext.	
					CER (%)	WER (%)	CER (%)	WER (%)	CER (%)	WER (%)
LSTM	ParisTech [San+16]	-	-	-	18,5	46,6	-	-	-	-
	LITIS [San+16]	✓	-	-	7,3	26,1	-	-	-	-
CNN + MDLSTM	A2IA [San+16]	✓	-	-	5,4	22,1	-	-	5,1	21,0
	RWTH [San+16]	✓	✓	-	4,8	20,9	-	-	-	-
CRNN	BYU [San+16]	-	-	-	5,1	21,1	-	-	-	-
seq2seq	CRNN + LSTM [Mic+19]	-	-	-	4,99	-	-	-	-	-
FCN	VAN (line level) [CCP22]	-	-	2,7M	**4,10**	**16,29**	-	-	-	-
Conv + encodeur Transformer	CFTNN	-	✓	3,5M	5,84	27,46	5,56	27,49	-	-
Conv + Transformer	VLT	-	✓	5,6M	6,59	28,12	4,97	24,23	-	-
Conv + Transformer	LT	-	✓	7,7M	6,14	27,32	5,30	25,26	-	-

plexe comme un Transformer n'est pas forcément nécessaire pour obtenir des résultats satisfaisants.

11.6.2 READ 2018

Dans cette section, nous comparons les résultats obtenus par nos approches avec les différentes méthodes à l'état de l'art sur le jeu de données READ 2018.

Nous rappelons que nos approches sont dans un premier temps entraînées sur les données génériques de READ 2018. De plus, nous utilisons des données synthétiques composées de tout type de contenus textuels disponibles, ainsi que des diverses augmentations et dégradations présentées dans la section 8.2 (p. 128). Pour les données génériques, l'ensemble d'apprentissage est utilisé dans un premier temps, avant d'y inclure l'ensemble de validation dans un second temps. Les architectures sont ensuite spécialisées sur les différents documents spécifiques en adaptant uniquement la partie encodeur lorsque cela s'applique. Enfin, les modèles sont évalués sur l'ensemble de test en utilisant la stratégie de prédiction.

L'architecture VLT ayant obtenu les meilleurs résultats dans les sections précédentes, nous proposons le modèle "VLT (Compétition)" dans le but de respecter strictement les instructions de la compétition. L'utilisation de données annotées extérieures ayant été proscrite, ce modèle est entraîné avec des données synthétiques générées uniquement à partir du contenu textuel de READ 2018. Cependant, nous pensons que l'utilisation de données synthétiques est bien moins coûteuse que l'annotation de données manuscrites réelles. Nous considérons donc que nos autres approches restent elles aussi dans le cadre de la compétition.

Dans la table 11.12, nous comparons ainsi les résultats de nos approches avec les autres de l'état de l'art [Str+18 ; YHM20 ; Sou+19]. En particulier, nous indiquons dans cette table les approches qui utilisent des modèles de langues additionnels (ParisTech, LITIS 2018, LITIS 2019 et PRHLT), ainsi que celles qui utilisent des augmentations en test (OSU, ParisTech et LITIS 2019). Enfin, il est aussi à noter que l'approche OSU n'a pas respecté les consignes pour la spécialisation des modèles sur les documents spécifiques. D'une part, l'ensemble des données spécifiques sont considérées comme annotées pour valider l'entraînement de leur architecture sur les données génériques. D'autre part, pour les spécialisations avec 1 et 4 pages annotées, les (15 et 12) pages restantes sont utilisées comme ensemble de validation. De même, l'approche ParisTech utilise une technique similaire pour valider leur architecture en utilisant une page, qui n'est pas incluse dans les

TABLE 11.12 – Comparaison avec les différentes approches de l'état de l'art sur le jeu de données READ 2018. Nous indiquons en gras les meilleurs résultats de l'état de l'art, et pour nos approches, celles améliorent les résultats existants. Nous indiquons de plus les approches qui utilisent un modèle de langue (LM), ainsi qu'une stratégie de prédiction (strat. pred.).

Type d'architecture	Méthode	LM	strat. pred.	CER (%) moyen par pages annotées				CER (%) moyen (0–16 pages) par document					CER (%) moyen
				0	1	4	16	Konzil. C	Schiller	Ricordi	Patzig	Schwerin	
MDLSTM + HMM	RPPDI [Str+18]	-	-	30,80	28,40	27,25	22,85	11,90	21,88	37,29	32,75	28,55	27,32
CRNN	OSU [Str+18] [a]	-	✓	31,39	17,73	13,27	9,02	9,39	21,10	23,27	23,17	12,98	17,86
	ParisTech [Str+18][a]	✓	✓	32,25	19,79	16,98	14,72	10,49	19,05	35,60	23,83	17,02	20,94
	LITIS 2018 [Str+18]	✓	-	35,29	22,51	16,89	11,34	9,14	25,69	30,50	25,18	18,04	21,51
	LITIS 2019 [Sou+19]	✓	✓	26,57	15,47	10,00	5,82	**5,94**	14,81	21,62	18,08	11,73	14,46
CRNN + BLSTM	PRHLT [Str+18]	✓	-	32,79	22,15	17,90	13,33	8,65	18,39	35,07	26,26	18,65	21,54
FCN	ASIUT [YHM20]	-	-	25,35	**12,63**	**8,28**	5,82	6,49	**13,77**	17,33	**14,85**	12,33	13,02
Conv + encodeur Transformer	CFTNN	-	✓	**25,06**	15,02	10,25	6,78	7,72	16,44	**15,27**	20,19	**11,37**	14,43
Conv + Transformer	VLT	-	✓	**24,28**	13,03	8,89	**5,64**	6,16	13,91	**15,21**	16,65	**11,71**	**12,96**
Conv + Transformer	LT	-	✓	25,74	13,33	8,84	**5,53**	6,61	14,44	**14,95**	18,02	**11,46**	13,36
Conv + Transformer	VLT (Compétition)	-	✓	**24,38**	13,07	9,19	**5,81**	6,29	13,87	**15,96**	17,24	**11,21**	13,11

a. Une comparaison équitable avec ces approche n'est pas possible comme des pages non incluses dans certains scénarios de spécification sont considérées comme annotées pour valider leur architecture.

données des scénarios 1 et 4 pages. Une comparaison équitable avec ces approches n'est donc pas possible.

Nos approches obtiennent d'excellents résultats sur le jeu de données READ 2018, avec un CER moyen de 14,43%, 12,96% et de 13,36% pour les architectures CFTNN, VLT et LT respectivement. En particulier, l'approche VLT obtient un meilleur CER moyen que celui de la meilleure approche de l'état de l'art [YHM20].

Comparaison par nombre de pages annotées

Avec 0 page annotée pour se spécialiser (table 11.12), c'est-à-dire en utilisant le modèle entraîné sur les données génériques, nos approches obtiennent d'excellents taux d'erreur. L'architecture LT se révèle être moins performante que les deux autres, à cause d'un manque de généralisation de l'architecture comme expliqué plus tôt. L'architecture CFTNN obtient un CER de 25,06% grâce à son faible nombre de paramètres, ce qui fait qu'elle dépasse les approches de l'état de l'art. Enfin, l'architecture VLT qui combine à la fois l'ajout d'un décodeur, et un faible nombre de paramètres, obtient un meilleur CER de 24,28%. Ces résultats indiquent ainsi que nos architectures sont capables de bien généraliser sur de nouvelles données en comparaison avec les autres approches. Cependant, les résultats obtenus sans adaptation restent élevés et ne peuvent pas être considérés comme suffisant.

En s'adaptant sur 1 et 4 pages annotées, nos approches obtiennent des résultats proches de ceux obtenus par la meilleure approche de l'état de l'art proposée par ASIUT [YHM20]. L'architecture FCN proposée dans cette approche semble démontrer de meilleures capacités d'adaptation avec très peu de données annotées. Elle possède moins de paramètres et est constituée de manière à être efficace avec peu de données. Avec peu de données annotées et la complexité de ces architectures, il est attendu que des architectures Transformer soient inférieures en comparaison. En revanche, il est à noter que nos architectures de Transformer légères ont des résultats très proches. Cela indique que nos approches ont d'excellentes capacités d'adaptation.

Enfin, avec le maximum de données annotées pour s'entraîner (16 pages), les architectures VLT et LT obtiennent des CER de 5,64% et 5,53% respectivement. Elles se placent donc devant les autres approches de l'état de l'art grâce à une quantité plus importante de données annotées pour les entraîner efficacement.

Dans la table 11.13, nous détaillons les résultats obtenus sur des spécification avec 16 pages annotées, notamment en précisant les résultats obtenus sur chaque document

TABLE 11.13 – Comparaison avec les approches de l'état de l'art sur les différents documents spécifiques de READ 2018 lorsque le maximum de pages annotées (16) est disponible pour réaliser une spécialisation. Nous indiquons en gras les meilleurs résultats de l'état de l'art, et pour nos approches, celles qui améliorent les résultats existants.

Méthode	CER (%) moyen par document spécifique avec 16 pages				
	Konzil. C	Schiller	Ricordi	Patzig	Schwerin
OSU [Str+18]	3,79	12,45	15,04	12,54	3,50
ParisTech [Str+18]	8,02	14,58	30,20	15,51	9,18
LITIS 2018 [Str+18]	4,81	19,57	16,37	12,83	6,61
PRHLT [Str+18]	4,98	12,55	28,52	16,35	7,12
RPPDI [Str+18]	9,18	16,29	30,49	28,30	24,90
ASIUT [YHM20]	2,83	8,17	11,44	**6,73**	**2,28**
LITIS 2019 [Sou+19]	**2,73**	8,41	9,72	7,19	2,74
CFTNN	3,42	9,19	**8,71**	10,38	2,93
VLT	2,95	**7,64**	**8,41**	7,67	2,63
LT	2,83	**7,92**	**8,09**	7,44	2,57
VLT (Compétition)	2,94	**7,80**	**8,62**	8,12	2,64

spécifique. Il est à noter que les architectures VLT et LT obtiennent un CER inférieur à 3% sur 2 des 5 documents spécifiques, tandis qu'elles obtiennent un taux d'erreur inférieur à 10% sur l'ensemble des documents spécifiques. Un taux d'erreur de 10% est en particulier considéré par la communauté comme un taux d'erreur acceptable pour des ajustements et corrections manuelles [Str+18].

Globalement, quel que soit le nombre de pages annotées, nos architectures démontrent de très bons résultats. Ces résultats indiquent en particulier les bonnes capacités de spécialisation avec peu de données annotées, ainsi que de bonnes capacités de généralisation pour nos architectures de Transformer légères.

Analyse par document spécifique

Concernant le CER moyen par document, nos approches obtiennent de meilleurs résultats que les approches de l'état de l'art sur 2 documents spécifiques sur 5. Elles sont particulièrement efficaces pour obtenir de faibles CER sur les documents spécifiques Ricordi (environ 15%) et Schwerin (environ 11,5%).

Pour le document spécifique Ricordi, nous rappelons que ce document est particulièrement complexe du fait qu'il est rédigé en italien, alors qu'aucun document italien n'est présent dans les données génériques. Nous observons en particulier un écart important (au moins 2 points de pourcentage de CER, soit 13% relativement) entre les résultats obtenus par nos approches et ceux de l'état de l'art. Selon nous, cet écart est principalement dû à un entraînement des modèles sur des données synthétiques au style visuel proche de celui du document spécifique (grâce à l'utilisation de dégradations, par exemple). Cela démontre ainsi que nos modèles sont capables de bien généraliser sur des données différentes, ici avec un nouveau langage.

Le contenu textuel utilisé pour générer des données synthétiques reste cependant très éloigné. Il pourrait être intéressant de considérer de l'italien dans les données synthétiques, ou encore d'autoriser la spécialisation du décodeur sur le document spécifique Ricordi. Cependant, le contenu textuel a été choisi en considérant avoir des connaissances uniquement sur les données génériques du jeu de données READ 2018. Concernant la spécialisation du décodeur sur Ricordi, nous avons ici choisi de garder une procédure uniforme par souci de clarté. Il pourrait cependant être intéressant d'étudier ces possibilités dans de futurs travaux.

Enfin, concernant les autres documents, nous observons des taux d'erreur très proches de ceux obtenus par les meilleurs approches [YHM20 ; Sou+19] sur les documents spécifiques Konzilsprotokolle C et Schiller. Nous observons cependant un écart important sur le document spécifique Patzig avec l'approche proposé par ASIUT, qui se place loin devant l'ensemble des autres approches.

De plus, comme le montre la table 11.13, nos architectures obtiennent d'excellents résultats lorsqu'elles sont entraînées avec 16 pages annotées. Sur les documents Schiller et Ricordi, les approches VLT et LT obtiennent de bien meilleurs résultats que ceux des approches de l'état de l'art. Sur le document spécifique Ricordi, nous remarquons là encore un écart important compris entre 1 et 2 points de pourcentage de CER (entre 10 et 20% relativement) avec l'ancienne meilleure approche. De même, les résultats restent proches de l'état de l'art sur les autres documents.

De manière globale, nos approches démontrent de très bonnes capacités d'adaptation sur l'ensemble des documents spécifiques. Cette efficacité est notamment due à des architectures légères ainsi qu'à une stratégie d'entraînement pour garder une architecture capable de très bien généraliser sur de nouvelles données.

Approche VLT (Compétition)

Dans les tables 11.12 et 11.13, les résultats obtenus par l'approche VLT (Compétition) sont globalement légèrement inférieurs à ceux obtenus par l'approche VLT. L'approche VLT (Compétition) obtient un CER moyen de 13,11% ce qui la place proche des résultats de la meilleure approche de l'état de l'art [YHM20].

Les résultats obtenus par l'approche VLT étant meilleurs, cela confirme que l'apport de contenus textuels extérieurs est bénéfique à l'entraînement d'architecture de Transformer. Cependant, il est à noter que pour les documents Schiller et tout particulièrement Schwerin, nous obtenons des résultats légérement meilleurs en utilisant uniquement du contenu textuel interne à READ 2018.

Détails des meilleurs résultats

Enfin, nous fournissons dans la table 11.14 le détail des résultats obtenus par notre meilleure architecture, c'est-à-dire l'architecture VLT, sur l'ensemble des scénarios de test (couple document spécifique et nombre de pages annotées disponibles). Nous les fournissons dans le but de permettre de futures comparaisons avec d'autres approches.

TABLE 11.14 – Détails des résultats obtenus par l'architecture VLT sur chaque scénario (un document spécifique et un nombre de pages annotées) sur le jeu de données READ 2018. Nous indiquons pour chaque scénario la moyenne ainsi que l'écart-type observé sur les 4 blocs de validation croisée.

Document	CER (%) ↓ moyen pour chaque scenario				
	0 page	1 page	4 pages	16 pages	Moyenne
Konzil. C	10,54	6,81 ± 0,73	4,33 ± 0,32	2,95 ± 0,12	6,16 ± 0,46
Schiller	21,20	14,92 ± 0,12	11,86 ± 0,26	7,64 ± 0,31	13,91 ± 0,24
Ricordi	22,96	16,72 ± 0,89	12,77 ± 0,13	8,41 ± 0,29	15,21 ± 0,54
Patzig	29,37	17,46 ± 0,60	12,10 ± 0,07	7,67 ± 0,23	16,65 ± 0,37
Schwerin	30,09	9,37 ± 0,70	4,75 ± 0,13	2,63 ± 0,08	11,71 ± 0,41
Moyenne	24,28	13,03 ± 0,66	8,89 ± 0,18	5,64 ± 0,22	12,96 ± 0,42

Synthèse des résultats sur READ 2018

Globalement, les approches proposées obtiennent d'excellents résultats. L'architecture VLT obtient le meilleur CER moyen de 12,96% sur le jeu de données READ 2018. De plus, nos architectures dépassent l'état de l'art sans adaptation, mais aussi avec le maximum de données annotées. Cela indique que nos architectures font preuve à la fois de généralisation et de bonnes capacités d'adaptation. Nos approches se montrent aussi capables d'obtenir les meilleurs résultats sur 2 des 5 documents spécifiques, et en particulier pour le document spécifique Ricordi qui est écrit en italien. Ceci confirme en particulier les capacités de nos architectures en termes d'adaptation et de généralisation. Enfin, sur les autres scénarios, nos approches obtiennent des résultats proches de ceux des meilleures approches.

11.7 Évaluation des prédictions des modèles

Dans cette section, nous abordons plus en profondeur les erreurs commises par nos meilleurs modèles. Concernant le jeu de données READ 2016, l'architecture VLT, entraînée en combinaison de données synthétiques paramétrées, est utilisée. Nos meilleurs modèles sur les jeux de données spécifiques de READ 2018 sont basés sur l'architecture VLT, et sont entraînés avec des données synthétiques. La procédure d'entraînement est utilisée, et nous étudions ici les modèles spécifiés aux données spécifiques de READ 2018 avec 16 pages d'annotations.

11.7.1 Répartition des erreurs

Dans la table 11.15, nous indiquons comment se répartissent les taux d'erreur obtenus par nos modèles selon les jeux de données considérés. Nous rappelons qu'un taux d'erreur inférieur à 10% est considéré par la communauté comme acceptable pour des corrections manuelles [Str+18].

Sur le jeu de données READ 2016, nous observons que 42,44% des images de ligne de texte sont correctement prédites, tandis que 19,33% obtiennent un taux d'erreur supérieur à 10%.

Sur les jeux de données spécifiques de READ 2018, nous obtenons des résultats assez variés. Les documents spécifiques Konzilprotokolle C et Schwerin, obtiennent en particulier de faibles taux d'erreur. Sur ces documents, 52,79% et 60,31% des lignes de texte sont prédites sans erreurs. Le nombre de taux d'erreurs supérieurs à 10% reste globale-

TABLE 11.15 – Répartition des CER sur les jeux de données READ 2016 et les différents jeux de données spécifiques de READ 2018. Pour les jeux de données spécifiques, nos architectures sont spécialisées avec 16 pages.

Jeux de données		CER moyen (%)	Répartition du CER			
			0%	0–5%	5–10%	>10%
READ 2016		4,97	42,44	21,44	16,78	19,33
READ 2018	Konzil. C	2,95	52,79	25,42	14,52	7,26
	Schiller	7,64	13,11	29,51	25,82	31,55
	Ricordi	8,41	22,50	20,71	26,79	30,00
	Patzig	7,67	18,06	22,40	28,12	31,42
	Schwerin	2,63	60,31	19,22	12,95	7,52

ment assez faible, et est selon nous raisonnable. En revanche, pour les autres documents, le nombre de prédictions obtenant au moins 10% d'erreurs reste très important. Cela concerne 31,55%, 30,00% et 31,42% des prédictions produites sur les jeux de données Schiller, Ricordi et Patzig respectivement.

Ces résultats indiquent en particulier, que de nombreux travaux sont encore possibles pour améliorer la qualité des prédictions sur des documents anciens.

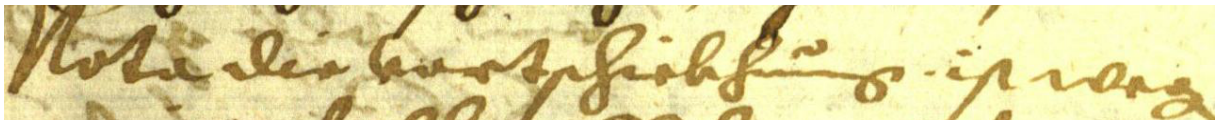
11.7.2 Analyse qualitative des prédictions

Dans la figure 11.2, nous montrons des résultats de prédiction sur le jeu de données READ 2016. Nous présentons en particulier des prédictions avec différentes quantités d'erreurs.

La figure 11.3 montre quant à elle, des prédictions sur les différents documents spécifiques, avec un nombre d'erreurs illustrant les taux d'erreur moyen obtenus par nos approches.

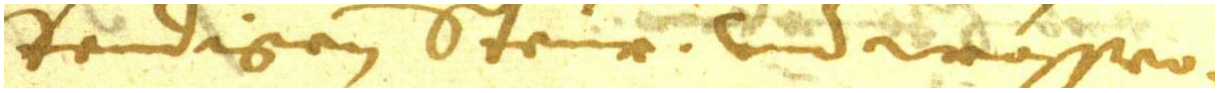
11.8 Discussion

Dans cette section, nous avons montré que les architectures proposées sont capables d'obtenir de faibles taux d'erreur sur des documents anciens, complexes et contenant peu de données annotées. Les architectures VLT et LT, incluant un décodeur Transformer,



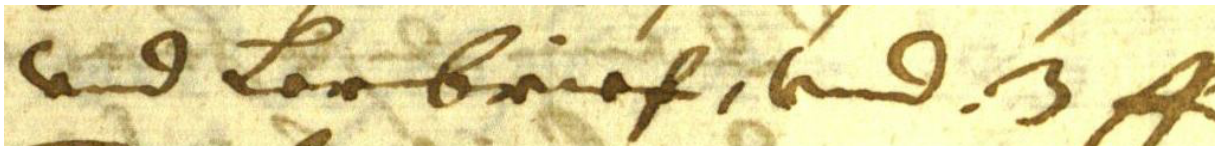
Vérité terrain : `Nota die vortschickhūng. ist weg`
Prédiction : `Nota die vortschickhūng. ist weg`

(a) Exemple de ligne correctement reconnue par notre modèle.



Vérité terrain : `stendigen Steūr. vnd Wasser¬`
Prédiction : `stendigen Steūr. vnd wasser¬`

(b) Exemple de prédiction comportant des erreurs. Sur cet exemple, le modèle VLT obtient un CER de 3,57% et un WER de 25,00%.



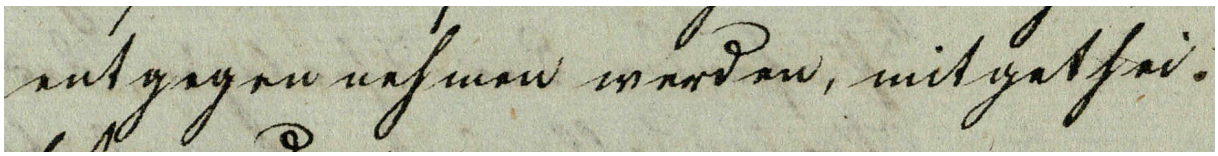
Vérité terrain : `vnd Lerbrief, vnd .3 fl`
Prédiction : `vnd Lernbrief, vnd [.]3 fl`

(c) Exemple de prédiction comportant des erreurs. Sur cet exemple, le modèle VLT obtient un CER de 8,70% et un WER de 40,00%.

Vérité terrain : `mūnd HūePscheren Annders¬`
Prédiction : `mūnd Zūepascherrn Annders¬`

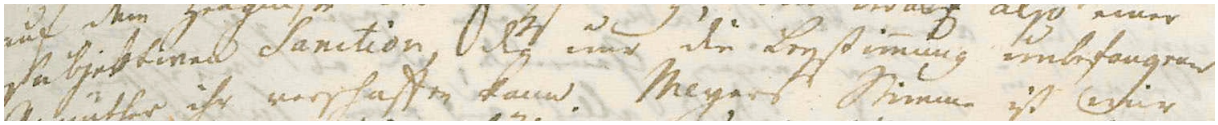
(d) Exemple de prédiction comportant des erreurs. Sur cet exemple, le modèle VLT obtient un CER de 16,00% et un WER de 33,33%.

FIGURE 11.2 – Exemples de prédictions sur des images du jeu de test de READ 2016. Les prédictions sont obtenues avec l'architecture VLT entraînée avec des données synthétiques paramétrées au style du jeu de données. L'utilisation de crochets "[...]" dénote des caractères manquants dans la prédiction.



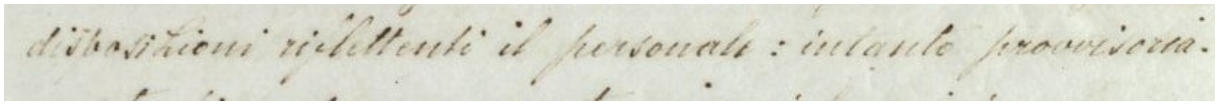
Vérité terrain : `entgegen nehmen werden, mitgethei¬`
Prédiction : `entgegen[_]nehmen werden, mitgethei¬`

(a) Exemple de prédiction sur le document spécifique Konzilprotokolle C. Sur cet exemple, le modèle VLT obtient un CER de 2,94% et un WER de 50,00%.



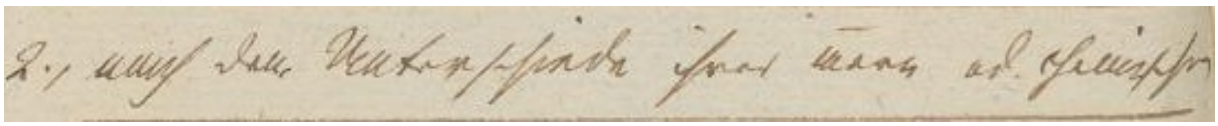
Vérité terrain : `subjektiven Sanction, die nur die Beystimung unbefangener`
Prédiction : `subjektiven Tanction, die nur die Beistimung unbefangen[er]`

(b) Exemple de prédiction sur le document spécifique Schiller. Cette image contient en particulier plusieurs lignes de texte. Sur cet exemple, le modèle VLT obtient un CER de 7,01% et un WER de 42,86%.



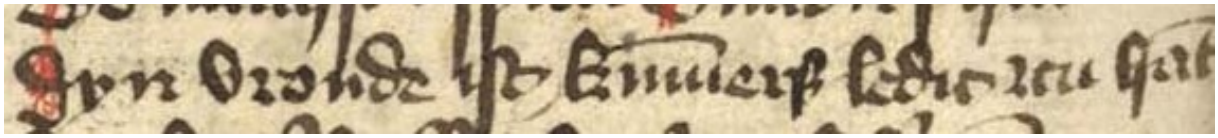
Vérité terrain : `disposizioni riflettenti il personale: intanto provvisoria¬`
Prédiction : `distrosizioni riglettenti il personale: intanto prov[v]isoria.`

(c) Exemple de prédiction sur le document spécifique Ricordi. Sur cet exemple, le modèle VLT obtient un CER de 8,47% et un WER de 50,00%.



Vérité terrain : `2., nach dem Unterschiede ihrer inern od. chemischen`
Prédiction : `2., nach dem Unterschiede ihrer in[ern] od. chemischer`

(d) Exemple de prédiction sur le document spécifique Patzig. Sur cet exemple, le modèle VLT obtient un CER de 7,69% et un WER de 25,00%.



Vérité terrain : `Dyn vroude ist kumersz ledic zcu hat`
Prédiction : `Dyn vroude ist kumersz ledic zcu hal`

(e) Exemple de prédiction sur le document spécifique Schwerin. Sur cet exemple, le modèle VLT obtient un CER de 2,78% et un WER de 14,29%.

FIGURE 11.3 – Exemples de prédictions sur des images du jeu de test de READ 2018. Les prédictions sont obtenues avec notre architecture VLT entraînée sur des données génériques et avec des données synthétiques. Elles sont ensuite spécialisées sur les différents documents spécifiques. Nous montrons des exemples représentatifs des taux d'erreur moyens obtenus sur ces jeux de données. L'utilisation de crochets "[...]" dénote des caractères manquants dans la prédiction.

peuvent modéliser la langue et réduire les taux d'erreur. De plus, comme elles restent assez légères en comparaison à des architectures de Transformer bien plus larges, elles ont l'avantage d'être entraînées sur peu de données annotées, et donc d'obtenir de bons résultats sur des données anciennes.

Nous avons utilisé diverses stratégies, dont l'intérêt a été validé dans cette section. Les données synthétiques au style des documents anciens ont été évaluées et se sont démontrées particulièrement utiles. Nous avons vu que leur premier intérêt réside dans le contenu textuel apporté, qui permet d'apprendre à modéliser efficacement la langue. Le style visuel reste lui aussi important, et en particulier pour des documents homogènes comme READ 2016. Nous avons aussi évalué l'impact de la stratégie d'entraînement proposée pour une spécialisation sur des jeux de données avec peu de données annotées disponibles. Elle s'est révélée être intéressante pour réduire efficacement les taux d'erreur, ainsi que la variance des résultats. La stratégie de prédiction s'est quant à elle à nouveau montrée efficace pour lisser d'éventuelles erreurs et réduire les taux d'erreur.

Sur le jeu de données READ 2016, l'architecture VLT obtient un excellent CER de 4,97 ce qui la place en deuxième position. Sur READ 2018, l'architecture VLT obtient un CER moyen de 12,96%, meilleur que ceux obtenus par les autres approches de l'état de l'art. Sans données annotées pour spécifier les modèles, notre architecture obtient les meilleurs résultats. Avec 16 pages annotées, elle obtient un CER moyen de 5,64% meilleur que les autres approches. De plus, elle obtient les meilleurs résultats sur 2 des 5 documents spécifiques. Sur les autres scénarios, nos architectures obtiennent des résultats proches des meilleures approches de l'état de l'art.

Conclusion générale

Résumé des contributions et des résultats

Dans ce manuscrit, nous avons cherché à appliquer des architectures Transformer pour la tâche de reconnaissance d'écriture manuscrite dans le but de réduire les taux d'erreur. Plus particulièrement, nous nous sommes penchés sur le problème des documents anciens. Cet objectif est complexe en raison du faible nombre de données annotées disponibles.

Nous avons alors proposé nos différentes contributions, à savoir :

— **des architectures de Transformer légères pour la reconnaissance d'écriture**, dans le but de s'entraîner avec peu de données annotées ;

— **deux algorithmes de génération de données synthétiques**, qui peuvent être utilisés pour générer des données modernes ou anciennes ;

— ainsi que **des stratégies d'entraînement et de prédiction** pour réduire les taux d'erreur de nos architectures.

Nous rappelons par ailleurs que nous mettons à disposition l'intégralité du code, des poids des modèles entraînés, ainsi que des paramètres utilisés lors des entraînements.

Nous avons appliqué avec succès nos approches sur des écritures manuscrites modernes, ainsi que sur des écritures anciennes. Nous résumons maintenant ces contributions, ainsi que quelques résultats marquants.

Architectures de Transformer légères pour la reconnaissance d'écriture manuscrite

Nous avons cherché à tirer parti des architectures Transformer et de leurs propriétés afin de réduire les taux d'erreur des approches existantes sur des documents anciens. Compte tenu des quantités de données annotées disponibles, et afin de simplifier les entraînements, nous avons choisi d'utiliser des architectures légères.

Dans le chapitre 6, nous avons proposé l'architecture convolutive à Transformer rapide (CFTNN). Elle reste légère et simple en combinant simplement des couches de convolutions et un encodeur Transformer. Nous avons validé ses performances dans le chapitre 10,

et nous avons montré qu'elle est capable de performances similaires à une architecture CRNN. Elle a l'avantage de permettre un processus d'inférence rapide, ce qui peut être utile dans des scénarios où la vitesse de prédiction est un facteur important. L'architecture CFTNN permet en particulier de traiter jusqu'à 60 pages d'écriture par seconde.

Dans le chapitre 7, nous avons considéré l'utilisation d'un décodeur Transformer pour ajouter des capacités de modélisation de la langue et permettre une réduction des taux d'erreur. Nous avons ainsi proposé l'architecture MLT, qui est légère et qui peut être modulée afin de s'adapter à différentes quantités de données annotées. Nous avons ainsi proposé deux variantes légères : les architectures LT et VLT. Sur des documents modernes, et sans données additionnelles, elles obtiennent des résultats au niveau de l'état de l'art.

Génération de données synthétiques

Pour permettre un apprentissage efficace et une réduction des taux d'erreur de nos architectures, nous utilisons des données synthétiques. Dans le chapitre 8, nous avons donc proposé en tant que contribution de la thèse, deux algorithmes de génération de données synthétiques.

Ils sont en particulier capables de générer des données synthétiques dans le style des documents anciens, ce qui n'est pas une tâche triviale. Nos algorithmes sont paramétrables, et permettent la génération de données dans des styles différents, comme ceux des écritures modernes, ou encore d'écritures imprimées. Ils peuvent aussi être utilisés pour générer des paragraphes entiers d'écritures manuscrites.

Dans la partie III, nous avons vu que les données synthétiques étaient cruciales pour entraîner efficacement des architectures Transformer, et en particulier pour des architectures avec un décodeur Transformer. Les résultats indiquent que nos architectures bénéficient grandement de l'apport d'un contenu textuel enrichi. L'aspect visuel des données synthétiques reste quand même important, surtout dans le cas de jeux de données homogènes.

Stratégies d'entraînement et de prédictions

Dans le chapitre 9, nous avons introduit en tant que contributions de la thèse deux stratégies affectant le processus d'entraînement et de prédiction.

Nous avons ainsi proposé une stratégie d'entraînement et de spécialisation sur des jeux de données spécifiques avec peu de données annotées. Elle permet en particulier à

nos approches de conserver un décodeur avec des capacités de modélisation de la langue génériques, et donc de limiter le surapprentissage du décodeur. L'encodeur est quant à lui entraîné à reconnaître le style visuel des données spécifiques.

D'autre part, nous avons proposé une stratégie de prédiction basée sur de l'augmentation en test, ainsi que sur un algorithme d'alignement et de vote des prédictions.

L'impact de ces stratégies a été évalué dans la partie III, et nous avons vu qu'elles permettent une réduction notable des taux d'erreur. Elles sont particulièrement utiles dans des scénarios où peu de données annotées sont disponible.

Bilan des résultats

En combinant les architectures proposées avec les différentes stratégies, nous avons vu que les approches proposées dans cette étude sont capables d'obtenir de faibles taux d'erreur sur des jeux de données usuels.

Taux d'erreur compétitifs Sur des jeux de données modernes, nos approches sont compétitives et obtiennent des taux d'erreur proches des meilleures approches. Sur le jeu de données d'écritures manuscrites modernes, IAM, et sans utiliser de données privées, l'architecture VLT obtient un excellent CER de 4,23%, ce qui la place en deuxième position juste derrière la meilleure approche. L'architecture LT obtient quant à elle un très bon CER de 2,51% sur le jeu de données d'écritures françaises, RIMES. Elle se place à la troisième position sans utiliser de données annotées externes.

Sur des données anciennes, elles ont démontré être capables d'obtenir de faibles taux d'erreur tout en s'entraînant avec peu de données annotées. Sur le jeu de données READ 2016, l'architecture VLT obtient par exemple un CER de 4,97%, ce qui la place à la deuxième position des architectures sans modèle de langue externe. Sur le jeu de données READ 2018, nos stratégies permettent à l'architecture VLT d'obtenir de faibles taux d'erreur avec très peu de pages annotées, malgré la complexité d'une architecture Transformer. Avec environ 500 lignes de textes pour se spécialiser, elle surpasse l'état de l'art et obtient des CER allant de 2,63% à 8,41% selon les documents. De plus, elle obtient les meilleures performances sur deux documents complexes, notamment un dans une langue non vue à l'entraînement.

Intérêts des architectures légères En comparaison à des architectures bien plus larges [Kan+22 ; Li+23], nos architectures montrent des résultats comparables, voire

meilleurs dans certains cas. Ces résultats indiquent en particulier qu'une architecture large n'est alors pas forcément nécessaire. Au contraire, une architecture légère et adaptée aux quantités de données disponibles, peut être largement suffisante.

L'utilisation d'architectures légères a différents avantages. D'une part, il est plus simple de faire converger des architectures légères en comparaison à des architectures plus larges. Elles sont en particulier moins sujet à des phénomènes de suraprentissage. De plus, les vitesses d'entraînement, ainsi que de prédiction, sont meilleures en comparaison, ce qui est un avantage notable. D'autre part, elles nécessitent moins de données annotées pour être entraînées efficacement. Comme nous l'avons vu dans le chapitre 10, elles peuvent obtenir des résultats au niveau de l'état l'art sans ajout de données annotées supplémentaires ou de données synthétiques. Des architectures plus larges obtiennent de moins bons résultats en comparaison.

Des architectures légères sont beaucoup moins coûteuses, que ce soit en termes de temps, de coût financier ou d'énergie nécessaire.

Empreinte carbone des travaux réalisés

Nos architectures légères ont une empreinte carbone plus légère par rapport à des architectures plus larges. En comparaison à l'architecture de modèle de langue large Bloom [Sca+22] (176 milliards de paramètres) qui a nécessité 13,27 tonnes kg éq. CO_2 après amortissement [LVL22], l'entraînement de l'architecture VLT nécessite seulement 2,48 kg éq. CO_2, soit 5 000 fois moins. Pour aller plus loin, nous proposons ici un bilan de l'empreinte carbone de la thèse. Elle est en revanche calculée uniquement à partir de la consommation en électricité, et n'inclut pas de coûts additionnels, tels que les coûts de fabrication.

Sur la durée totale de la thèse, ainsi que du stage de recherche qui l'a précédée, les différentes expériences réalisées ont nécessité un total de 57 361 heures GPU réparties sur plus de 1 500 travaux soumis. Cela inclut les entraînements de nos modèles ainsi que de nombreuses expériences qui ont été nécessaires pour converger vers les versions actuelles des architectures proposées dans ce manuscrit.

En nous basant sur un taux d'utilisation des cartes graphiques à 100% de leur capacité de manière constante, en négligeant l'empreinte de calculs réalisés par le processeur, et sur la base des chiffres mis à disposition [1], un total de 27 648 kWh a alors été consommé

1. `http://www.idris.fr/jean-zay/calcul-empreinte-carbone.html`

pour réaliser les différentes expériences. Le centre de calcul Jean Zay disposant d'une revalorisation de l'énergie perdue sous forme de chaleur, on peut considérer qu'un total de 14 856 kWh ont été utilisés pour entraîner nos architectures, après amortissement.

1 kWh produit en France correspond à environ 57g éq. CO_2 [LVL22] (7 fois moins qu'un pays tel que les États-Unis), notamment grâce à une énergie fortement décarbonée. Au total, cette thèse a eu une empreinte carbone de 846 kg éq. CO_2, ou 225 kg éq. CO_2 par année.

À titre de comparaison, cette empreinte carbone correspond environ à :

— une place aller simple de Paris à New-York ;

— 900 km de voiture par an (ou 2,5 km par jour) ;

— 5% de l'empreinte carbone annuelle moyenne d'un français [2] ;

— 4 kg de viande bovine par an ;

— 4 tasses de café par jour.

Perspectives et travaux futurs

Nos travaux offrent de multiples perspectives, et nous proposons ici plusieurs axes de recherches.

Adaptation avec plus de données annotées

Dans ce manuscrit, nous avons abordé une tâche d'entraînement générique, suivie d'une phase de spécialisation sur des documents spécifiques. Nous avons en particulier étudié l'impact de la quantité de données annotées pour spécialiser nos modèles, et nous avons montré que nos architectures légères sont adaptées pour de faibles quantités de données annotées.

Afin de minimiser la quantité d'annotations requises sur des documents spécifiques, il semble intéressant d'augmenter la quantité de données annotées disponibles lors de l'entraînement générique. Pour cela, la combinaison de jeux de données existants peut être considérée. La taille du jeu d'entraînement générique, et son influence sur les performances en spécialisation pourraient alors être étudiées.

2. https://www.statistiques.developpement-durable.gouv.fr/sites/default/files/2020-01/datalab-essentiel-204-l-empreinte-carbone-des-francais-reste-%20stable-janvier2020.pdf

Diverses questions de recherche se posent alors, concernant les capacités d'apprentissage des architectures, ainsi que leurs capacités de généralisation. Il est alors intéressant de se demander quelles tailles d'architectures sont adaptées pour bénéficier d'un entraînement générique significatif, tout en s'adaptant à des documents spécifiques avec des données annotées limitées. Des architectures plus larges pourraient alors être utilisées. Cependant, une taille trop importante pourrait avoir un impact négatif sur les capacités de généralisation.

Afin de contrôler efficacement la taille de l'architecture, l'architecture modulaire que nous proposons pourrait être utilisée. De plus, notre stratégie d'entraînement et d'adaptation à des quantités limitées de données pourrait être bénéfique pour aider lors de la spécialisation de l'architecture.

Automatisation du processus de génération des données synthétiques pour des documents spécifiques

Nous avons proposé des approches pour générer de l'écriture synthétique, que nous utilisons notamment lors de la phase d'entraînement générique. Afin de diminuer le besoin en annotations sur des données spécifiques, il pourrait être intéressant de générer des données synthétiques dans le style visuel de ces données.

Alors que nous avons manuellement sélectionné les polices d'écriture, ce choix peut découler en des polices non-représentatives des écritures spécifiques. Il peut alors être intéressant de quantifier de manière automatique la proximité d'une police d'écriture à des écritures réelles. Cela permettrait alors de choisir des polices d'écriture pertinentes. De même, un processus similaire pourrait être appliqué pour déterminer des contenus textuels adaptés, ainsi que les augmentations à utiliser.

Pour ce faire, il pourrait être intéressant de considérer uniquement les images d'écritures du document spécifiques, qui n'aurait pas besoin d'être annotées.

Reconnaissance de paragraphes et de pages avec peu de données annotées

Les travaux de cette thèse se sont focalisés sur la reconnaissance de lignes de texte isolées. Cependant, l'emploi de lignes isolées peut restreindre l'accès à des informations contextuelles pertinentes. En effet, la segmentation en lignes peut rompre le contexte au sein d'une phrase ou d'un paragraphe. L'utilisation de ces informations contextuelles

pourrait se révéler particulièrement bénéfique pour améliorer la reconnaissance de certains mots, comme le démontrent notamment les résultats présentés par Coquenet et al. [CCP23].

Pour prendre en compte les informations présentes dans d'autres lignes, une première approche consisterait à mettre bout à bout des lignes consécutives. Cela permettrait ainsi d'ajouter des informations supplémentaires à des systèmes de reconnaissance d'écriture existants, comme nos approches. Les architectures Transformer seraient capables de traiter efficacement de telles séquences étendues, facilitant ainsi l'incorporation d'informations pertinentes pour la reconnaissance des caractères. Cependant, il convient de noter que l'entraînement de ces architectures deviendrait nettement plus complexe en raison de la longueur accrue des séquences.

Une deuxième approche consisterait à traiter des pages entières, en apprenant l'ordre de lecture. Les approches au niveau des pages nous semblent être pertinentes, performantes, et bien plus adaptées pour de futurs travaux dans le domaine. Cela permet en particulier de supprimer le besoin d'une étape de segmentation en ligne, qui peut parfois entraîner des erreurs.

Néanmoins, les approches qui traitent des pages sont plus complexes à entraîner que des approches qui traitent des lignes. En effet, il est important d'apprendre l'ordre de lecture, et implicitement, la structure du document. Il est alors crucial de les entraîner efficacement avec suffisamment de données. Une perspective de recherche intéressante serait d'exploiter nos approches pour réduire le besoin en données annotées dans le contexte des méthodes axées sur les pages. En particulier, nous avons développé un algorithme de génération de données synthétiques capable de générer des paragraphes entiers. Il pourrait alors être utilisé pour aider l'entraînement de ces approches. Nos architectures légères pourraient quant à elle servir pour la reconnaissance de pages, combinées avec un encodeur capable de traiter des pages entières.

Grâce à l'intégration de connaissances supplémentaires sur des lignes consécutives, un paragraphe, ou une page complète, nos approches pourraient obtenir des taux d'erreurs réduits. De plus, nos approches pourraient être utilisées pour traiter d'autres tâches avec peu de changements, comme par exemple la reconnaissance d'entités nommées.

Extension à des domaines connexes

Dans ce manuscrit, nous avons proposé diverses architectures et stratégies dans le but de réduire le besoin en données annotées pour la tâche de reconnaissance d'écriture manuscrite.

Avec peu de données annotées, l'entraînement d'architectures de Transformer reste relativement complexe, et en particulier pour des architectures larges. Pourtant, dans d'autres domaines, des modèles ont été entraînés sur des quantités de données massives. C'est en particulier le cas des modèles de langue larges. Ces modèles de langue larges se sont avérés performants en apprenant des caractéristiques riches et génériques, et en modélisant efficacement le contexte. Ils pourraient fortement bénéficier à des architectures de reconnaissance d'écriture en améliorant leur capacité de modélisation contextuelle.

Les modèles de langue larges peuvent être intégrés directement dans des architectures de reconnaissance d'écriture en tant que décodeur. Cependant, une adaptation est nécessaire, car les modèles de langues traditionnels opèrent au niveau des mots, tandis que les systèmes de reconnaissance d'écriture travaillent au niveau des caractères. Toutefois, le réentraînement de ces modèles peut être une tâche complexe compte tenu du nombre important de paramètres, ainsi que du faible nombre de données annotées. Il pourrait alors être intéressant de proposer des stratégies de réapprentissage avec peu de données annotées.

À l'inverse, nos approches pourraient être adaptées à d'autres domaines connexes, surtout lorsque les quantités de données annotées sont limitées. Cela pourrait concerner des domaines tels que la reconnaissance de texte dans des scènes d'images naturelles ; l'extraction d'information ; ou encore la reconnaissance de la parole.

Vers un apprentissage sans annotations

Dans cette thèse, nous avons cherché à entraîner des architectures avec peu de données. Nous avons vu que nos approches peuvent obtenir des taux d'erreur proches de 5% sur des documents anciens avec seulement 16 pages annotées (environ 500 lignes de textes isolées). Cependant, il peut être intéressant de diminuer encore plus les besoins en données annotées.

Afin de réduire le besoin en données annotées, il peut être intéressant de considérer des processus d'apprentissage qui considèrent des données non annotées. Ces données existent en d'importantes quantités, et sont facilement accessibles.

Des techniques d'auto-apprentissage (*Self-Supervised Learning*) pourraient être utilisées dans le but d'obtenir des modèles optiques génériques et performants. Cela permettrait notamment d'entraîner un modèle générique surpassant ceux obtenus par des méthodes d'apprentissage supervisé classiques.

Une tâche de complétion de zones masquées dans l'image (ou d'*inpainting*) pourrait être réalisée, à la manière de ce qui peut être utilisé pour le traitement automatique de la langue [Dev+18] ou en reconnaissance d'images [Bao+21]. L'avantage de cette tâche est qu'elle n'exige pas d'annotations sur les données d'entraînement.

En masquant des zones de la taille des caractères, le modèle apprend ainsi spontanément diverses caractéristiques et informations structurelles relatives à l'écriture. À l'inverse, un modèle supervisé a tendance à se concentrer sur les caractéristiques spécifiques aux classes à reconnaître. Cela lui permet de mieux généraliser et de s'adapter plus facilement à de nouveaux concepts, à la manière de l'apprentissage humain.

De plus, pour réduire le besoin en données annotées, l'exploration de techniques d'apprentissage semi-supervisé est une autre piste prometteuse. Ces méthodes tirent parti d'exemples non étiquetés pendant le processus d'apprentissage, souvent en utilisant des pseudo-étiquettes attribuées à des exemples non supervisés. Notamment, des techniques telles que le co-apprentissage pourraient être employées pour atteindre cet objectif.

BIBLIOGRAPHIE

Publications personnelles

Communication dans un congrès international avec publication des actes et comité de relecture

[Bar+22b] Killian BARRERE et al., « A Light Transformer-Based Architecture for Handwritten Text Recognition », in : *International Workshop on Document Analysis Systems*, Springer, 2022, p. 275-290.

Communication dans un congrès international sans actes

[Bar+21b] Killian BARRERE et al., « Transformers for Historical Handwritten Text Recognition », in : *Doctoral Consortium - ICDAR 2021*, prix du meilleur poster, Nibal Nayef and Jean-Christophe Burie, Lausanne, Switzerland, sept. 2021, URL : `https://hal.science/hal-03485262`.

Communication dans des congrès francophones sans actes

[Bar+21a] Killian BARRERE et al., « Architecture Légère de Transformer pour la Reconnaissance d' Écriture Manuscrite », in : *Symposium International Francophone sur l'Ecrit et le Document (SIFED) 2021*, Lyon, France, déc. 2021, URL : `https://hal.science/hal-03857807`.

[Bar+22a] Killian BARRERE et al., « A Light Transformer-Based Architecture for Handwritten Text Recognition », in : *Reconnaissance des Formes, Image, Apprentissage et Perception (RFIAP) 2022*, Vannes, France, juill. 2022, URL : `https://hal.science/hal-03857723`.

[Bar+22c] Killian BARRERE et al., *Architecture Transformer Légère pour la Reconnaissance de Textes Anciens*, SIFED 2022 - Symposium International Francophone sur l'Ecrit et le Document, Poster, oct. 2022, URL : `https://hal.science/hal-03857509`.

[Bar+23a] Killian Barrere et al., *Entraînement d'architectures type Transformer sur peu de données : application à la reconnaissance de texte manuscrit ancien*, SIFED 2023 - Symposium International Francophone sur l'Écrit et le Document, Poster, Camille Kurtz and Florence Cloppet and Nicole Vincent, juin 2023, URL : https://hal.science/hal-04129144.

Article de journal international en cours de relecture

[Bar+23b] Killian Barrere et al., « Training Transformer Architectures on Few Annotated Data : an Application to Historical Handwritten Text Recognition », in : *International Journal on Document Analysis and Recognition (IJDAR)* (2023).

Réferences

[Aug+06] Emmanuel Augustin et al., « RIMES evaluation campaign for handwritten mail processing », in : *International Workshop on Frontiers in Handwriting Recognition (IWFHR'06)*, 2006, p. 231-235.

[AY01] Nafiz Arica et Fatos T Yarman-Vural, « An overview of character recognition focused on off-line handwriting », in : *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)* 31.*2* (2001), p. 216-233.

[Bah+16] Dzmitry Bahdanau et al., « End-to-end attention-based large vocabulary speech recognition », in : *ICASSP 2016*, IEEE, 2016, p. 4945-4949.

[Bao+21] Hangbo Bao et al., « Beit : Bert pre-training of image transformers », in : *arXiv preprint arXiv :2106.08254* (2021).

[BCB14] Dzmitry Bahdanau, Kyunghyun Cho et Yoshua Bengio, « Neural machine translation by jointly learning to align and translate », in : *arXiv preprint arXiv :1409.0473* (2014).

[Bhu+21] Ankan Kumar Bhunia et al., « Handwriting transformers », in : *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2021, p. 1086-1094.

[BKH16] Jimmy Lei Ba, Jamie Ryan Kiros et Geoffrey E Hinton, « Layer normalization », in : *arXiv preprint arXiv :1607.06450* (2016).

[BLM17] Théodore Bluche, Jérôome Louradour et Ronaldo Messina, « Scan, attend and read : End-to-end handwritten paragraph recognition with mdlstm attention », in : *14th IAPR ICDAR*, t. 1, IEEE, 2017, p. 1050-1055.

[Blu+14] Théodore Bluche et al., « The a2ia arabic handwritten text recognition system at the open hart2013 evaluation », in : *2014 11th IAPR International Workshop on Document Analysis Systems*, IEEE, 2014, p. 161-165.

[Blu16] Théodore Bluche, « Joint line segmentation and transcription for end-to-end handwritten paragraph recognition », in : *NIPS*, 2016, p. 838-846.

[BM17] Théodore Bluche et Ronaldo Messina, « Gated convolutional recurrent neural networks for multilingual handwriting recognition », in : *14th IAPR ICDAR*, IEEE, 2017, p. 646-651.

[Bra+16] James Bradbury et al., « Quasi-recurrent neural networks », in : *arXiv preprint arXiv :1611.01576* (2016).

[CCP21] Denis Coquenet, Clément Chatelain et Thierry Paquet, « SPAN : a Simple Predict & Align Network for Handwritten Paragraph Recognition », in : *ICDAR*, Springer, 2021, p. 70-84.

[CCP22] Denis Coquenet, Clément Chatelain et Thierry Paquet, « End-to-end handwritten paragraph text recognition using a vertical attention network », in : *IEEE Transactions on Pattern Analysis and Machine Intelligence* 45.*1* (2022), p. 508-524.

[CCP23] Denis Coquenet, Clément Chatelain et Thierry Paquet, « DAN : a Segmentation-free Document Attention Network for Handwritten Document Recognition », in : *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2023), doi : 10.1109/TPAMI.2023.3235826.

[Cha+16] William Chan et al., « Listen, attend and spell : A neural network for large vocabulary conversational speech recognition », in : *2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, IEEE, 2016, p. 4960-4964.

[Chi+18] Chung-Cheng Chiu et al., « State-of-the-art speech recognition with sequence-to-sequence models », in : *ICASSP*, IEEE, 2018, p. 4774-4778.

[Cho+14] Kyunghyun Cho et al., « Learning phrase representations using RNN encoder-decoder for statistical machine translation », in : *arXiv preprint arXiv :1406.1078* (2014).

[Coq+19] Denis Coquenet et al., « Have convolutions already made recurrence obsolete for unconstrained handwritten text recognition ? », in : *ICDAR Workshop on Machine Learning*, t. 5, IEEE, 2019, p. 65-70.

[dAr+22] Rafael d'Arce et al., « Self-attention Networks for Non-recurrent Handwritten Text Recognition », in : *Frontiers in Handwriting Recognition : 18th International Conference, ICFHR 2022, Hyderabad, India, December 4–7, 2022, Proceedings*, Springer, 2022, p. 389-403.

[Dav+20] Brian Davis et al., « Text and style conditioned gan for generation of offline handwriting lines », in : *BMVC*, 2020.

[Dev+18] Jacob Devlin et al., « Bert : Pre-training of deep bidirectional transformers for language understanding », in : *arXiv preprint arXiv :1810.04805* (2018).

[Dia+21] Daniel Hernandez Diaz et al., « Rethinking text line recognition models », in : *arXiv preprint arXiv :2104.07787* (2021).

[DKN14] P. Doetsch, M. Kozielski et H. Ney, « Fast and Robust Training of Recurrent Neural Networks for Offline Handwriting Recognition », in : *2014 14th International Conference on Frontiers in Handwriting Recognition*, sept. 2014, p. 279-284, DOI : 10.1109/ICFHR.2014.54.

[Dos+20] Alexey Dosovitskiy et al., « An image is worth 16x16 words : Transformers for image recognition at scale », in : *arXiv preprint arXiv :2010.11929* (2020).

[Dut+18] Kartik Dutta et al., « Improving CNN-RNN hybrid networks for handwriting recognition », in : *2018 16th International Conference on Frontiers in Handwriting Recognition (ICFHR)*, IEEE, 2018, p. 80-85.

[DZN16] Patrick Doetsch, Albert Zeyer et Hermann Ney, « Bidirectional decoder networks for attention-based end-to-end offline handwriting recognition », in : *15th ICFHR*, 2016, p. 361-366.

[Fis97] Jonathan G Fiscus, « A post-processing system to yield reduced word error rates : Recognizer output voting error reduction (ROVER) », in : *1997 IEEE Workshop on Automatic Speech Recognition and Understanding Proceedings*, IEEE, 1997, p. 347-354.

[FK61] W. Nelson Francis et Henry Kučera, *Brown University Standard Corpus of Present-Day American English (Brown corpus)*, 1961, URL : https://www.nltk.org/book/ch02.html#brown-corpus (visité le 23/06/2023).

[FL90] Scott E Fahlman et Christian Lebiere, « Learning to execute », in : *Carnegie Mellon University, School of Computer Science, Technical Report* CMU-CS-90-100 (1990).

[Fog+20] Sharon Fogel et al., « Scrabblegan : Semi-supervised varying length handwritten text generation », in : *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2020, p. 4324-4333.

[Fri+12] Volkmar Frinken et al., « Long-short term memory neural networks language modeling for handwriting recognition », in : *Proceedings of the 21st International Conference on Pattern Recognition (ICPR2012)*, IEEE, 2012, p. 701-704.

[FU15] Volkmar Frinken et Seiichi Uchida, « Deep BLSTM neural networks for unconstrained continuous handwritten text recognition », in : *13th ICDAR*, IEEE, 2015, p. 911-915.

[GB10] Xavier Glorot et Yoshua Bengio, « Understanding the difficulty of training deep feedforward neural networks », in : *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, 2010, p. 249-256.

[GE11] Emmanuele Grosicki et Haikal El-Abed, « Icdar 2011-french handwriting recognition competition », in : *2011 International Conference on Document Analysis and Recognition*, IEEE, 2011, p. 1459-1463.

[GFS05] Alex Graves, Santiago Fernández et Jürgen Schmidhuber, « Bidirectional LSTM networks for improved phoneme classification and recognition », in : *International Conference on Artificial Neural Networks*, Springer, 2005, p. 799-804.

[GFS07] Alex Graves, Santiago Fernández et Jürgen Schmidhuber, « Multidimensional recurrent neural networks », in : *International conference on artificial neural networks*, Springer, 2007, p. 549-558.

[GJM13] Alex Graves, Navdeep Jaitly et Abdel-rahman Mohamed, « Hybrid speech recognition with deep bidirectional LSTM », in : *2013 IEEE workshop on automatic speech recognition and understanding*, IEEE, 2013, p. 273-278.

[Gra+06] Alex Graves et al., « Connectionist temporal classification : labelling unsegmented sequence data with recurrent neural networks », in : *23rd international conference on Machine learning*, ACM, 2006, p. 369-376.

[Gra+08] Alex Graves et al., « A novel connectionist system for unconstrained handwriting recognition », in : *IEEE transactions on pattern analysis and machine intelligence* 31.*5* (2008), p. 855-868.

[GS09] Alex Graves et Jürgen Schmidhuber, « Offline handwriting recognition with multidimensional recurrent neural networks », in : *NIPS*, 2009, p. 545-552.

[GSC00] Felix A Gers, Jürgen Schmidhuber et Fred Cummins, « Learning to forget : Continual prediction with LSTM », in : *Neural computation* 12.*10* (2000), p. 2451-2471.

[Gu+17] Jiatao Gu et al., « Non-autoregressive neural machine translation », in : *International Conference on Learning Representations (ICLR)*, 2017.

[GW21] Ji Gan et Weiqiang Wang, « HiGAN : Handwriting imitation conditioned on arbitrary-length texts and disentangled styles », in : *Proceedings of the AAAI Conference on Artificial Intelligence*, t. 35, 9, 2021, p. 7484-7492.

[He+16] Kaiming He et al., « Deep residual learning for image recognition », in : *CVPR*, 2016, p. 770-778.

[Hin+12] Geoffrey E. Hinton et al., « Improving neural networks by preventing co-adaptation of feature detectors », in : *arXiv preprint arXiv :1207.0580* (2012).

[HS97] Sepp Hochreiter et Jürgen Schmidhuber, « Long short-term memory », in : *Neural computation* 9.*8* (1997), p. 1735-1780.

[Hua+17] Gao Huang et al., « Densely connected convolutional networks », in : *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, p. 4700-4708.

[Ing+19] R Reeve Ingle et al., « A Scalable Handwritten Text Recognition System », in : *15th IAPR ICDAR*, IEEE, 2019.

[IS15] Sergey Ioffe et Christian Szegedy, « Batch normalization : Accelerating deep network training by reducing internal covariate shift », in : *International conference on machine learning*, PMLR, 2015, p. 448-456.

[Jou+17] Nicholas Journet et al., « Doccreator : A new software for creating synthetic ground-truthed document images », in : *Journal of imaging* 3.*4* (2017), p. 62.

[Kan+22] Lei Kang et al., « Pay attention to what you read : non-recurrent handwritten text-line recognition », in : *Pattern Recognition* 129 (2022), p. 108766.

[KB14] Diederik Kingma et Jimmy Ba, « Adam : A Method for Stochastic Optimization », in : *International Conference on Learning Representations* (déc. 2014).

[KDN13] Michał Kozielski, Patrick Doetsch et Hermann Ney, « Improvements in rwth's system for off-line handwriting recognition », in : *Document Analysis and Recognition (ICDAR), 2013 12th International Conference on*, IEEE, 2013, p. 935-939.

[KJ16] Praveen Krishnan et CV Jawahar, « Generating synthetic data for text recognition », in : *arXiv preprint arXiv :1608.04224* (2016).

[Kol+21] Alexander Kolesnikov et al., « An Image is Worth 16x16 Words : Transformers for Image Recognition at Scale », in : *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*, OpenReview.net, 2021.

[KPZ17] Woo-Young Kang, Kyung-Wha Park et Byoung-Tak Zhang, « Extremely Sparse Deep Learning Using Inception Modules with Dropfilters », in : *14th ICDAR*, t. 1, IEEE, 2017, p. 448-453.

[KSH12] Alex Krizhevsky, Ilya Sutskever et Geoffrey E Hinton, « Imagenet classification with deep convolutional neural networks », in : *NIPS*, 2012, p. 1097-1105.

[KW+52] Jack Kiefer, Jacob Wolfowitz et al., « Stochastic estimation of the maximum of a regression function », in : *The Annals of Mathematical Statistics* 23.*3* (1952), p. 462-466.

[LB05] Marcus Liwicki et Horst Bunke, « IAM-OnDB-an on-line English sentence database acquired from handwritten text on a whiteboard », in : *Eighth International Conference on Document Analysis and Recognition (ICDAR'05)*, IEEE, 2005, p. 956-961.

[LeC+89] Yann LeCun et al., « Backpropagation applied to handwritten zip code recognition », in : *Neural computation* 1.*4* (1989), p. 541-551.

[LeC+98] Yann LeCun et al., « Gradient-based learning applied to document recognition », in : *Proceedings of the IEEE* 86.*11* (1998), p. 2278-2324.

[Lev+66] Vladimir I Levenshtein et al., « Binary codes capable of correcting deletions, insertions, and reversals », in : *Soviet physics doklady*, t. 10, 8, Soviet Union, 1966, p. 707-710.

[Li+23] Minghao Li et al., « TrOCR : Transformer-based Optical Character Recognition with Pre-trained Models », in : *AAAI 2023*, fév. 2023.

[Liu+19] Yinhan Liu et al., « Roberta : A robustly optimized bert pretraining approach », in : *arXiv preprint arXiv :1907.11692* (2019).

[Liw+07] Marcus Liwicki et al., « A novel approach to on-line handwriting recognition based on bidirectional long short-term memory networks », in : *Proceedings of the 9th International Conference on Document Analysis and Recognition, ICDAR 2007*, 2007.

[LK14] Jérôme Louradour et Christopher Kermorvant, « Curriculum learning for handwritten text line recognition », in : *2014 11th IAPR International Workshop on Document Analysis Systems*, IEEE, 2014, p. 56-60.

[LVL22] Alexandra Sasha Luccioni, Sylvain Viguier et Anne-Laure Ligozat, « Estimating the carbon footprint of bloom, a 176b parameter language model », in : *arXiv preprint arXiv :2211.02001* (2022).

[LWH15] Qi Liu, Lijuan Wang et Qiang Huo, « A study on effects of implicit and explicit language model information for DBLSTM-CTC based handwriting recognition », in : *2015 13th International Conference on Document Analysis and Recognition (ICDAR)*, IEEE, 2015, p. 461-465.

[Mad+22] Boraq Madi et al., « HST-GAN : Historical Style Transfer GAN for Generating Historical Text Images », in : *International Workshop on Document Analysis Systems*, Springer, 2022, p. 523-537.

[MB02] U-V Marti et Horst Bunke, « The IAM-database : an English sentence database for offline handwriting recognition », in : *IJDAR* 5.*1* (2002), p. 39-46.

[MB95] Nelson Morgan et Hervé Bourlard, « An introduction to hybrid HMM/connectionist continuous speech recognition », in : *IEEE Signal Processing Magazine* 12.*3* (1995), p. 25-42.

[Mer+16] Stephen Merity et al., « Pointer sentinel mixture models », in : *arXiv preprint arXiv :1609.07843* (2016).

[MHN13] Andrew L. Maas, Awni Y. Hannun et Andrew Y. Ng, « Rectifier Nonlinearities Improve Neural Network Acoustic Models », in : *Proceedings of the 30th International Conference on Machine Learning (ICML-13)*, JMLR Workshop et Conference Proceedings, 2013, p. 3-11.

[Mic+19] Johannes Michael et al., « Evaluating sequence-to-sequence models for handwritten text recognition », in : *2019 International Conference on Document Analysis and Recognition (ICDAR)*, IEEE, 2019, p. 1286-1293.

[ML15] Ronaldo Messina et Jerome Louradour, « Segmentation-free handwritten Chinese text recognition with LSTM-RNN », in : *2015 13th International Conference on Document Analysis and Recognition (ICDAR)*, IEEE, 2015, p. 171-175.

[MM19] Tsvetomila Mihaylova et André FT Martins, « Scheduled sampling for transformers », in : *arXiv preprint arXiv :1906.07651* (2019).

[Moy+14] Bastien Moysset et al., « The A2iA multi-lingual text recognition system at the second Maurdor evaluation », in : *2014 14th International Conference on Frontiers in Handwriting Recognition*, IEEE, 2014, p. 297-302.

[Nar] Sean Naren, *PyTorch bindings for Warp-ctc*, `https://github.com/SeanNaren/warp-ctc`.

[OJ04] Kyoung-Su Oh et Keechul Jung, « GPU implementation of neural networks », in : *Pattern Recognition* 37.*6* (2004), p. 1311-1314.

[Par+18] Niki Parmar et al., « Image transformer », in : *International conference on machine learning*, PMLR, 2018, p. 4055-4064.

[Pas+19] Adam Paszke et al., *PyTorch : An Imperative Style, High-Performance Deep Learning Library*, 2019, arXiv : `1912.01703 [cs.LG]`.

[Pen+19] Dezhi Peng et al., « A Fast and Accurate Fully Convolutional Network for End-to-End Handwritten Chinese Text Segmentation and Recognition », in : *15th IAPR ICDAR*, IEEE, 2019.

[Pha+14] Vu Pham et al., « Dropout improves recurrent neural networks for handwriting recognition », in : *14th ICFHR*, IEEE, 2014, p. 285-290.

[PMV16] Joan Puigcerver, Daniel Martin-Albo et Mauricio Villegas, *Laia : A deep learning toolkit for htr*, 2016.

[Ptu+19] Raymond Ptucha et al., « Intelligent character recognition using fully convolutional neural networks », in : *Pattern Recognition* 88 (2019), p. 604-613.

[Pui17] Joan Puigcerver, « Are multidimensional recurrent layers really necessary for handwritten text recognition ? », in : *14th IAPR ICDAR*, t. 1, IEEE, 2017, p. 67-72.

[Res] Baidu Research, *warp-ctc : A fast parallel implementation of CTC, on both CPU and GPU*, `https://github.com/baidu-research/warp-ctc`.

[Reu+22] Christian Reul et al., « Open Source Handwritten Text Recognition on Medieval Manuscripts Using Mixed Models and Document-Specific Finetuning », in : *International Workshop on Document Analysis Systems*, Springer, 2022, p. 414-428.

[Ria+22] Nauman Riaz et al., « Conv-transformer architecture for unconstrained offline Urdu handwriting recognition », in : *International Journal on Document Analysis and Recognition (IJDAR)* 25.*4* (2022), p. 373-384.

[RM51] Herbert Robbins et Sutton Monro, « A stochastic approximation method », in : *The annals of mathematical statistics* (1951), p. 400-407.

[RMN09] Rajat Raina, Anand Madhavan et Andrew Y Ng, « Large-scale deep unsupervised learning using graphics processors », in : *Proceedings of the 26th annual international conference on machine learning*, 2009, p. 873-880.

[Rom+13] Verónica Romero et al., « The ESPOSALLES database : An ancient marriage license corpus for off-line handwriting recognition », in : *Pattern Recognition* 46.*6* (2013), p. 1658-1669.

[RRC15] Anupama Ray, Sai Rajeswar et Santanu Chaudhury, « Text recognition using deep blstm networks », in : *2015 eighth international conference on advances in pattern recognition (ICAPR)*, IEEE, 2015, p. 1-6.

[San+16] Joan Andreu Sanchez et al., « ICFHR2016 competition on handwritten text recognition on the READ dataset », in : *2016 15th International Conference on Frontiers in Handwriting Recognition (ICFHR)*, IEEE, 2016, p. 630-635.

[San+17] Joan Andreu Sanchez et al., « ICDAR2017 competition on handwritten text recognition on the READ dataset », in : *2017 14th IAPR international conference on document analysis and recognition (ICDAR)*, t. 1, IEEE, 2017, p. 1383-1388.

[SBY16] Baoguang Shi, Xiang Bai et Cong Yao, « An end-to-end trainable neural network for image-based sequence recognition and its application to scene text recognition », in : *IEEE transactions on pattern analysis and machine intelligence* 39.*11* (2016), p. 2298-2304.

[Sca+22] Teven Le Scao et al., « Bloom : A 176b-parameter open-access multilingual language model », in : *arXiv preprint arXiv :2211.05100* (2022).

[SCP17] Bruno Stuner, Clément Chatelain et Thierry Paquet, « LV-ROVER : lexicon verified recognizer output voting error reduction », in : *arXiv preprint arXiv :1707.07432* (2017).

[SDN16] Dewi Suryani, Patrick Doetsch et Hermann Ney, « On the benefits of convolutional neural network combinations in offline handwriting recognition », in : *15th ICFHR*, IEEE, 2016, p. 193-198.

[SGS15] Rupesh Kumar Srivastava, Klaus Greff et Jürgen Schmidhuber, « Highway networks », in : *arXiv preprint arXiv :1505.00387* (2015).

[SK21] Sumeet S Singh et Sergey Karayev, « Full Page Handwriting Recognition via Image to Sequence Extraction », in : *ICDAR*, Springer, 2021, p. 55-69.

[SLG78] Johansson Stig, Geoffrey N Leech et Helen Goodluck, « Manual of information to accompany the Lancaster-Oslo : Bergen Corpus of British English, for use with digital computers », in : *(No Title)* (1978).

[Sou+19] Yann Soullard et al., « Improving text recognition using optical and language model writer adaptation », in : *2019 International Conference on Document Analysis and Recognition (ICDAR)*, IEEE, 2019, p. 1175-1180.

[Sou+20] Yann Soullard et al., « Multi-scale Gated Fully Convolutional DenseNets for semantic labeling of historical newspaper images », in : *Pattern Recognition Letters* 131 (2020), p. 435-441.

[SP97] Mike Schuster et Kuldip K Paliwal, « Bidirectional recurrent neural networks », in : *IEEE Transactions on Signal Processing* 45.*11* (1997), p. 2673-2681.

[SS19] Martin Schall et Marc-Peter Schambach, « Dissecting Multi-line Handwriting for Multi-dimensional Connectionist Classification », in : *2019 15th IAPR International Conference on Document Analysis and Recognition (ICDAR)*, IEEE, 2019.

[SSP+03] Patrice Y Simard, David Steinkraus, John C Platt et al., « Best practices for convolutional neural networks applied to visual document analysis », in : *Icdar*, t. 3, 2003, Edinburgh, 2003.

[SSP19] Wassim Swaileh, Yann Soullard et Thierry Paquet, « A unified multilingual handwriting recognition system using multigrams sub-lexical units », in : *Pattern Recognition Letters* 121 (2019), p. 68-76.

[Str+18] Tobias Strauss et al., « ICFHR2018 competition on automated text recognition on a READ dataset », in : *2018 16th International Conference on Frontiers in Handwriting Recognition (ICFHR)*, IEEE, 2018, p. 477-482.

[Sun+17] Li Sun et al., « GMU : A Novel RNN Neuron and Its Application to Handwriting Recognition », in : *14th ICDAR*, t. 1, IEEE, 2017, p. 1062-1067.

[Swa+17] Wassim Swaileh et al., « Handwriting recognition with multigrams », in : *2017 14th IAPR International Conference on Document Analysis and Recognition (ICDAR)*, t. 1, IEEE, 2017, p. 137-142.

[SZ14] Karen Simonyan et Andrew Zisserman, « Very deep convolutional networks for large-scale image recognition », in : *arXiv preprint arXiv :1409.1556* (2014).

[Sze+16] Christian Szegedy et al., « Rethinking the inception architecture for computer vision », in : *CVPR*, 2016, p. 2818-2826.

[Tom+15] Jonathan Tompson et al., « Efficient object localization using convolutional networks », in : *CVPR*, 2015, p. 648-656.

[Tou+21] Hugo Touvron et al., « Training data-efficient image transformers & distillation through attention », in : *International conference on machine learning*, PMLR, 2021, p. 10347-10357.

[TW19] Chris Tensmeyer et Curtis Wigington, « Training full-page handwritten text recognition models without annotated line breaks », in : *2019 15th IAPR International Conference on Document Analysis and Recognition (ICDAR)*, IEEE, 2019.

[Vas+17] Ashish Vaswani et al., « Attention is all you need », in : *NIPS*, 2017, p. 5998-6008.

[VDN16] Paul Voigtlaender, Patrick Doetsch et Hermann Ney, « Handwriting recognition with large multidimensional long short-term memory recurrent neural networks », in : *15th ICFHR*, IEEE, 2016, p. 228-233.

[Vit67] Andrew Viterbi, « Error bounds for convolutional codes and an asymptotically optimum decoding algorithm », in : *IEEE transactions on Information Theory* 13.*2* (1967), p. 260-269.

[Vög+21] Lars Vögtlin et al., « Generating synthetic handwritten historical documents with OCR constrained GANs », in : *International Conference on Document Analysis and Recognition*, Springer, 2021, p. 610-625.

[Wan+17] Fei Wang et al., « Residual attention network for image classification », in : *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017, p. 3156-3164.

[Wan+20] Wenhui Wang et al., « Minilm : Deep self-attention distillation for task-agnostic compression of pre-trained transformers », in : *Advances in Neural Information Processing Systems* 33 (2020), p. 5776-5788.

[WF74] Robert A Wagner et Michael J Fischer, « The string-to-string correction problem », in : *Journal of the ACM (JACM)* 21.*1* (1974), p. 168-173.

[WH17] Jianfeng Wang et Xiaolin Hu, « Gated recurrent convolution neural network for ocr », in : *Advances in Neural Information Processing Systems* 30 (2017).

[Wig+17] Curtis Wigington et al., « Data augmentation for recognition of handwritten words and lines using a CNN-LSTM network », in : *2017 14th IAPR international conference on document analysis and recognition (ICDAR)*, t. 1, IEEE, 2017, p. 639-645.

[Wil] Essen Wilson, *PyTorch edit-distance functions*, `https://github.com/gmyofustc/pytorch-edit-distance`.

[WYL17] Yi-Chao Wu, Fei Yin et Cheng-Lin Liu, « Improving handwritten Chinese text recognition using neural network language models and convolutional neural network shape models », in : *Pattern Recognition* 65 (2017), p. 251-264.

[WZG21] Christoph Wick, Jochen Zöllner et Tobias Grüning, « Transformer for Handwritten Text Recognition Using Bidirectional Post-decoding », in : *ICDAR*, Springer, 2021, p. 112-126.

[WZG22] Christoph Wick, Jochen Zöllner et Tobias Grüning, « Rescoring sequence-to-sequence models for text line recognition with CTC-prefixes », in : *International Workshop on Document Analysis Systems*, Springer, 2022, p. 260-274.

[Xia+19] Shanyu Xiao et al., « Deep Network with Pixel-Level Rectification and Robust Training for Handwriting Recognition », in : *2019 15th IAPR International Conference on Document Analysis and Recognition (ICDAR)*, IEEE, 2019.

[Xie+17] Saining Xie et al., « Aggregated residual transformations for deep neural networks », in : *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, p. 1492-1500.

[Xio+18] Wayne Xiong et al., « The Microsoft 2017 conversational speech recognition system », in : *2018 IEEE international conference on acoustics, speech and signal processing (ICASSP)*, IEEE, 2018, p. 5934-5938.

[Xio+21] Yunyang Xiong et al., « Mobiledets : Searching for object detection architectures for mobile accelerators », in : *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2021, p. 3825-3834.

[Xu+15] Kelvin Xu et al., « Show, attend and tell : Neural image caption generation with visual attention », in : *International conference on machine learning*, 2015, p. 2048-2057.

[YB20] Mohamed Yousef et Tom E Bishop, « Origaminet : Weakly-supervised, segmentation-free, one-step, full page text recognition by learning to unfold », in : *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, p. 14710-14719.

[YHM20] Mohamed Yousef, Khaled F Hussain et Usama S Mohammed, « Accurate, data-efficient, unconstrained text recognition with convolutional neural networks », in : *Pattern Recognition* 108 (2020), p. 107482.

[Zam+14] Francisco Zamora-Martinez et al., « Neural network language models for off-line handwriting recognition », in : *Pattern Recognition* 47.*4* (2014), p. 1642-1652.

[Zha+17] Jianshu Zhang et al., « Watch, attend and parse : An end-to-end neural network based approach to handwritten mathematical expression recognition », in : *Pattern Recognition* 71 (2017), p. 196-206.

[Zha+20] Qian Zhang et al., « Transformer transducer : A streamable speech recognition model with transformer encoders and rnn-t loss », in : *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, IEEE, 2020, p. 7829-7833.

Annexe A

Visuels additionnels obtenus avec nos approches de génération de données synthétiques

Nous proposons ici différents visuels supplémentaires, dans différents scénarios, dans le but de démontrer les capacités de nos algorithmes de génération de donnés synthétiques.

L'ensemble des visuels décrits ci-dessous sont présentés dans les figures suivantes :

— **Figure A.1** : Lignes d'écritures anglaises modernes

— **Figure A.2** : Lignes d'écriture françaises modernes

— **Figure A.3** : Lignes d'écritures anciennes et génériques

— **Figure A.4** : Lignes d'écritures anciennes dans le style de READ 2016

— **Figure A.5** : Paragraphes anciens et génériques

— **Figure A.6** : Des paragraphes anciens avec une écriture française

— **Figure A.7** : Paragraphe anciens dans le style de READ 2016

— **Figure A.8** : Paragraphe modernes obtenus avec notre approche

Pour plus de détails sur les algorithmes de génération que nous proposons, nous invitons les lecteurs à se référer au chapitre 8 (p. 119).

Nous rappelons que nous mettons à disposition un répertoire gitlab public concernant les algorithmes de générations présentés : `https://gitlab.inria.fr/intuidoc-public/synthetic-handwriting-generation`. Ce répertoire gitlab contient en particulier :

— le code des algorithmes de génération de données synthétiques ;

— les polices d'écritures utilisées et leur poids respectif, dans différents scénarios ;

— les paramètres d'augmentations dans les divers scénarios proposés ;

— diverses images de fond.

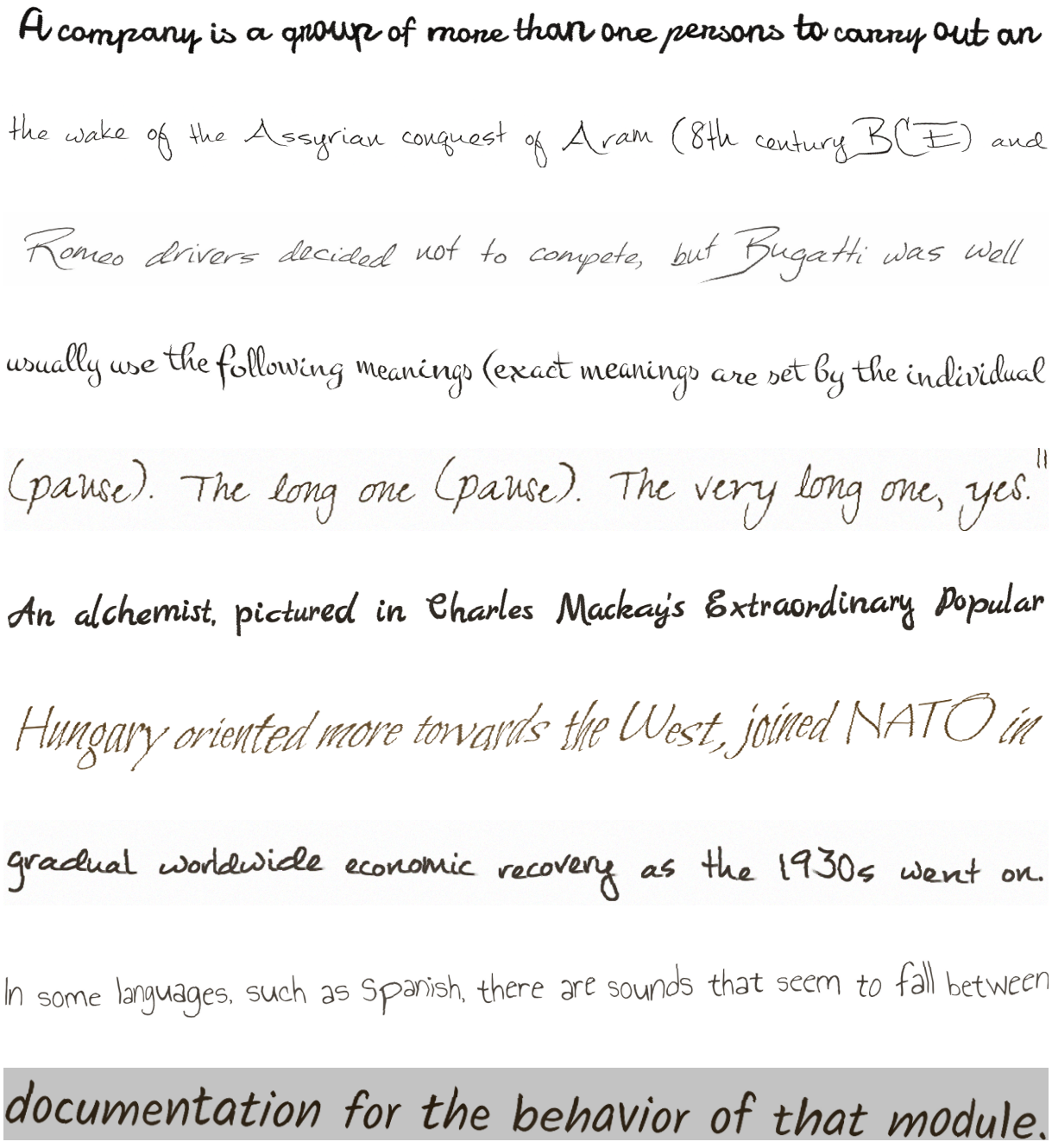


FIGURE A.1 – Données synthétiques dans le style d'écritures modernes obtenues par notre approche de génération. Ces données sont utilisées pour entraîner nos modèles sur le jeu de données IAM 2.1.1 (p. 30).

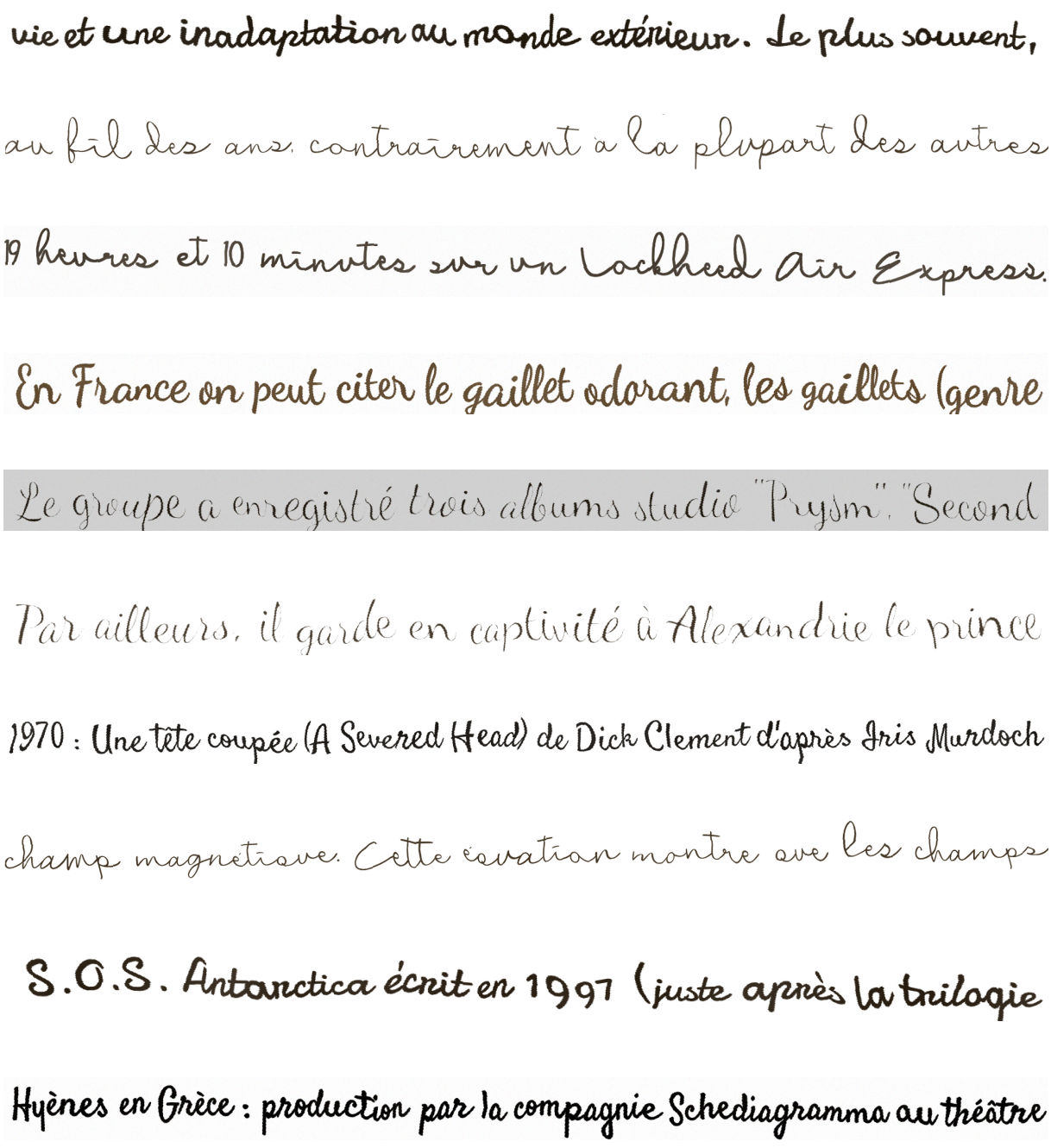


FIGURE A.2 – Exemples de données synthétiques dans le style d'écritures françaises modernes. Ces lignes servent en particulier à entraîner nos modèles sur le jeu de données RIMES, présenté dans la section 2.1.2 (p. 31).

FIGURE A.3 – Plusieurs exemples de lignes synthétiques dans le style de documents anciens générées avec notre approche. Ces images de lignes synthétiques sont en particulier obtenues avec les mêmes paramètres que ceux utilisés lors de nos expériences sur le jeu de données READ 2018 (*c.f.* la section 5.1.2 p. 76, ainsi que le chapitre 11 p. 189).

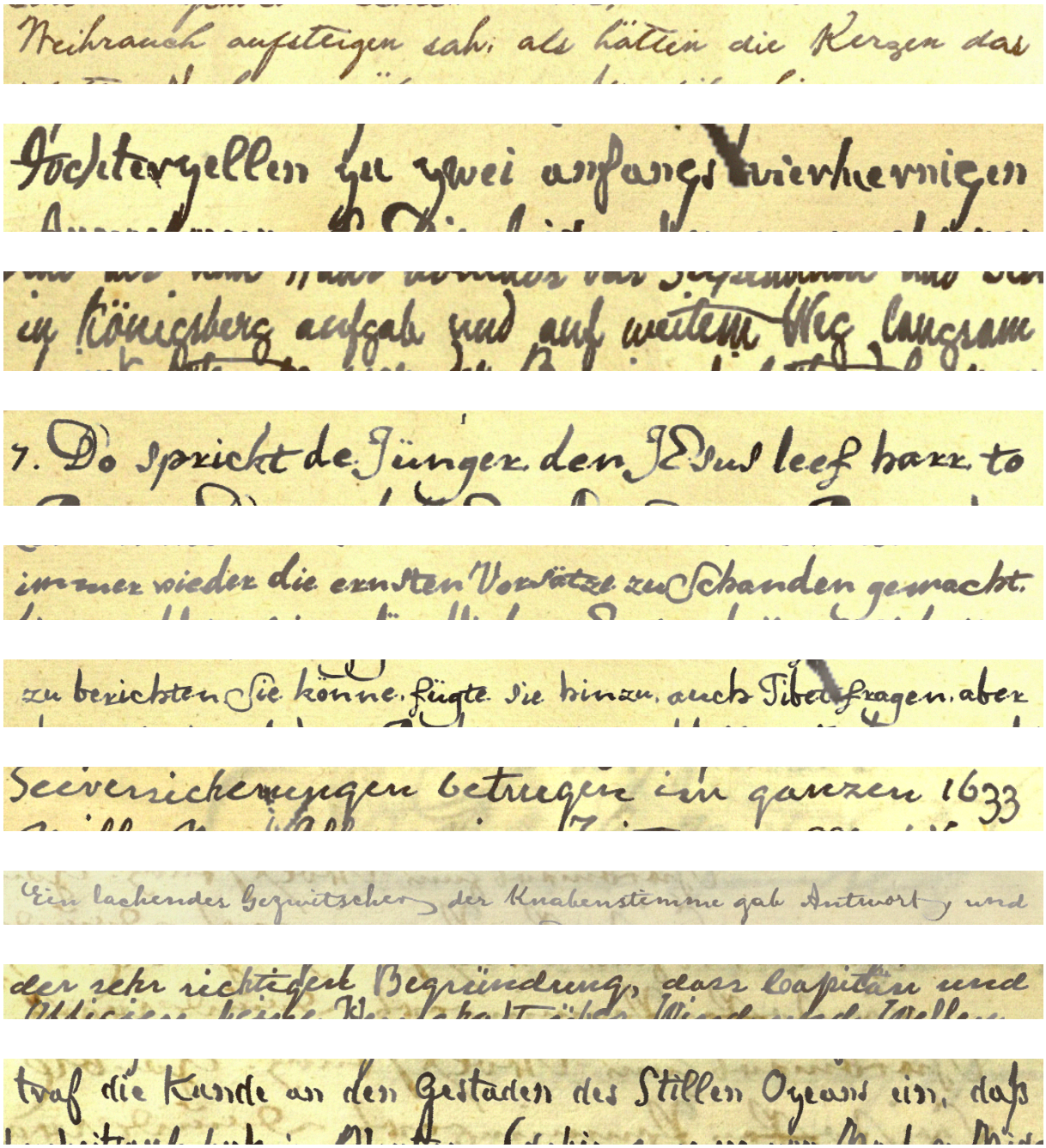


FIGURE A.4 – De multiples exemples de lignes de texte synthétiques, générées dans le but de ressembler au jeu de données READ 2016 présenté dans la section 5.1.2 (p. 74).

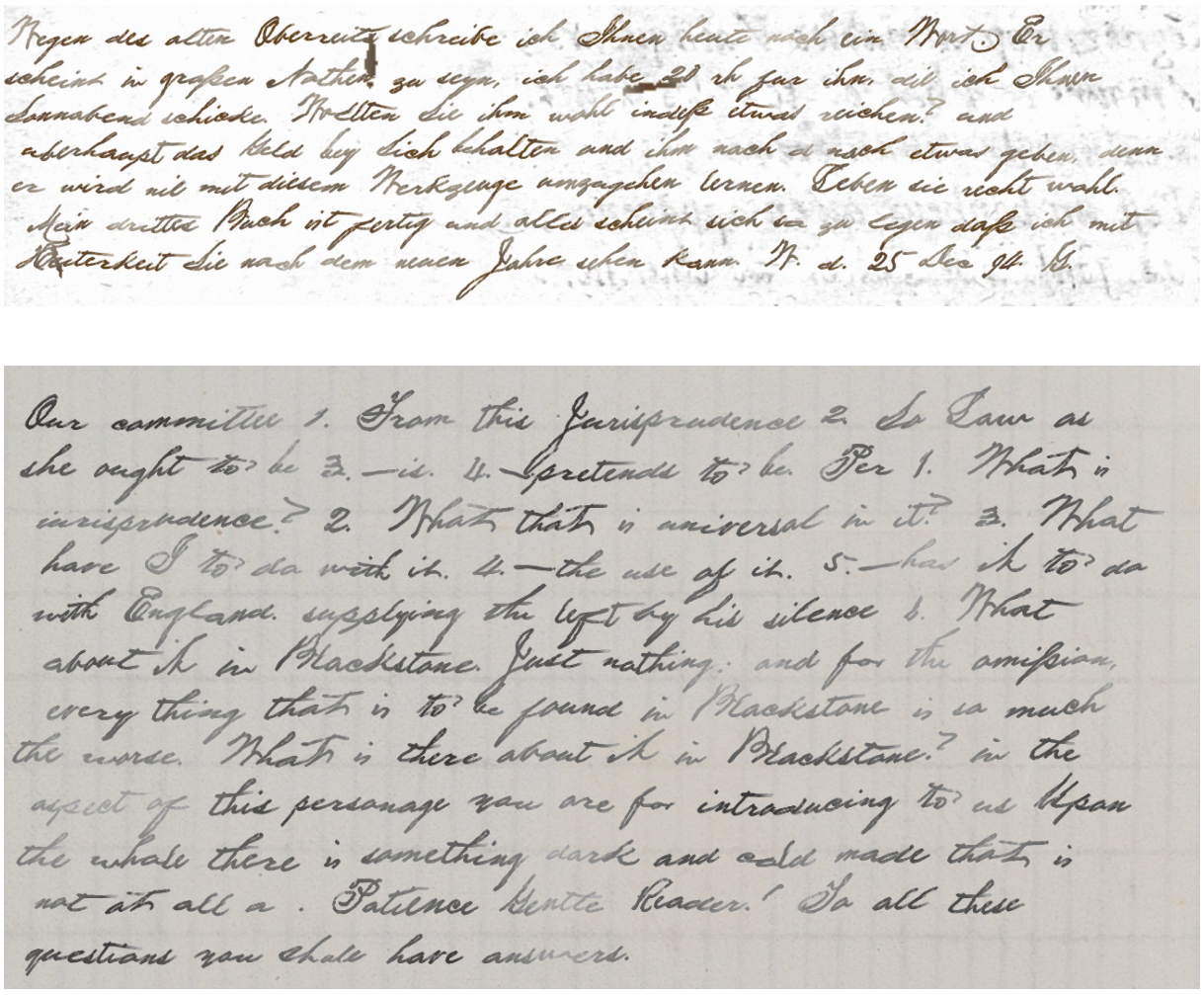


FIGURE A.5 – Exemples de paragraphes complets générés avec notre approche. Ici, nous visons à générer des paragraphes dans un style ancien et générique.

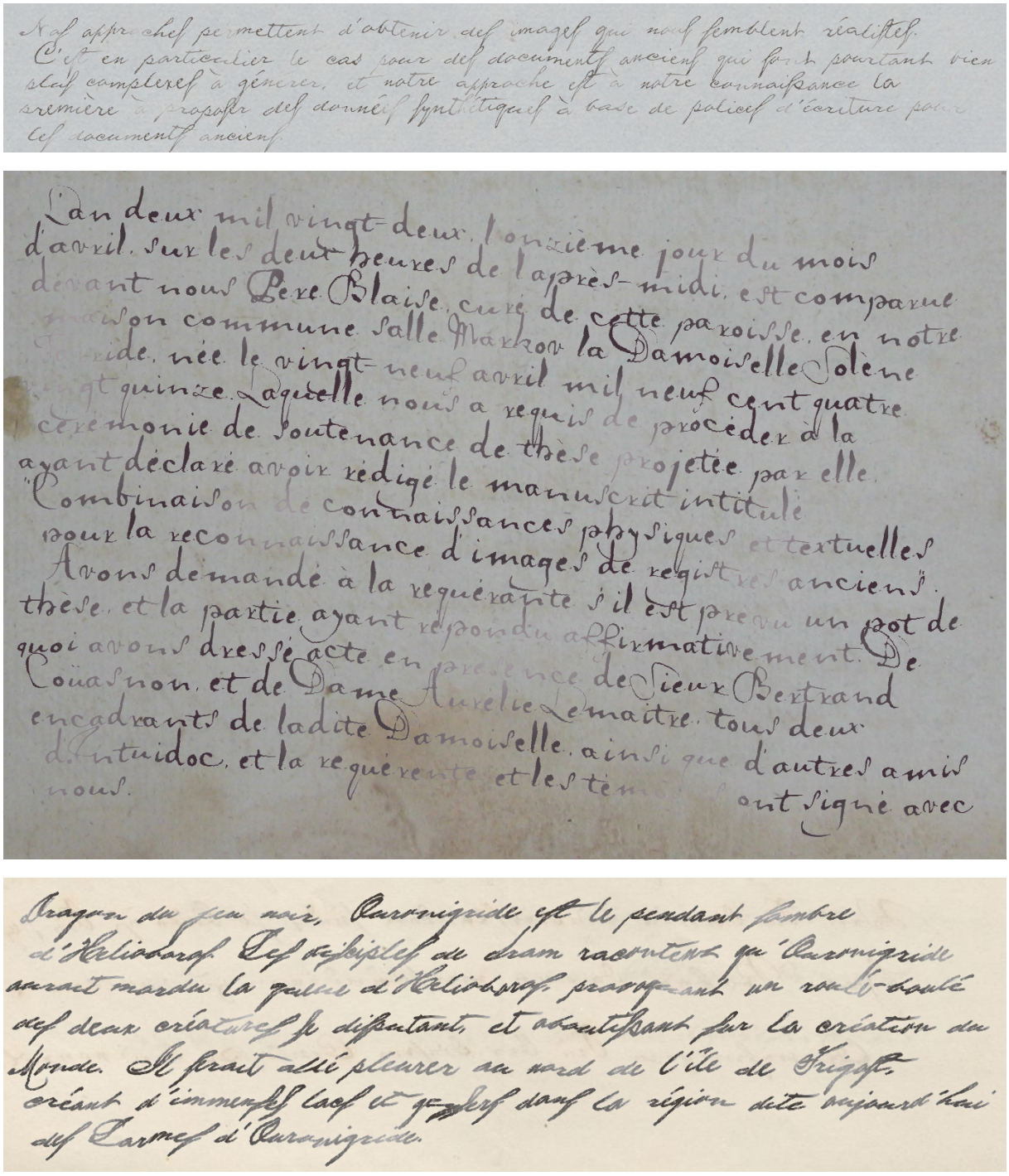


FIGURE A.6 – Des paragraphes d'écriture française générés dans un style ancien.

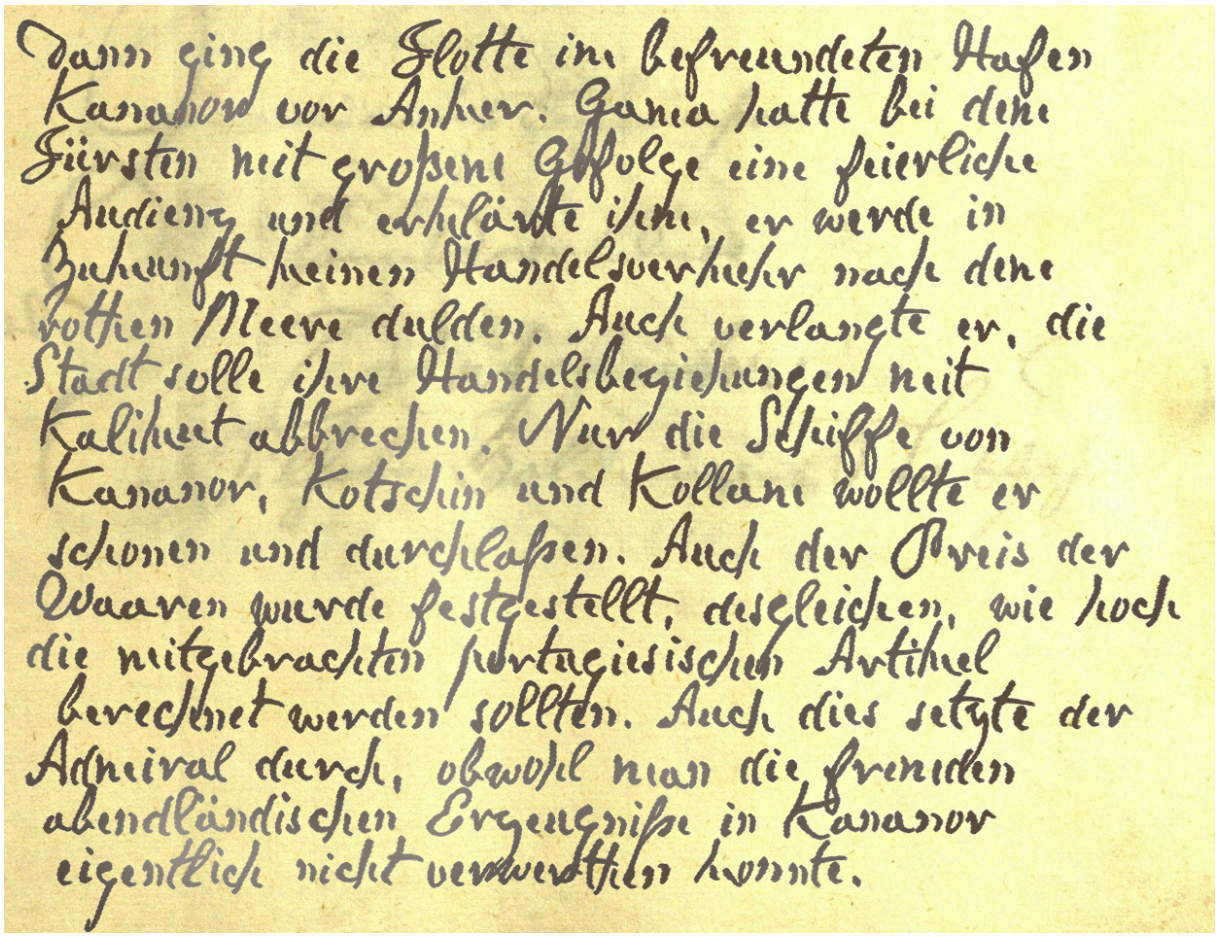


Figure A.7 – Paragraphe synthétique généré avec notre approche dans le but d'obtenir des données proches visuellement du jeu de données READ 2016 (*c.f.* section 5.1.2, p. 74).

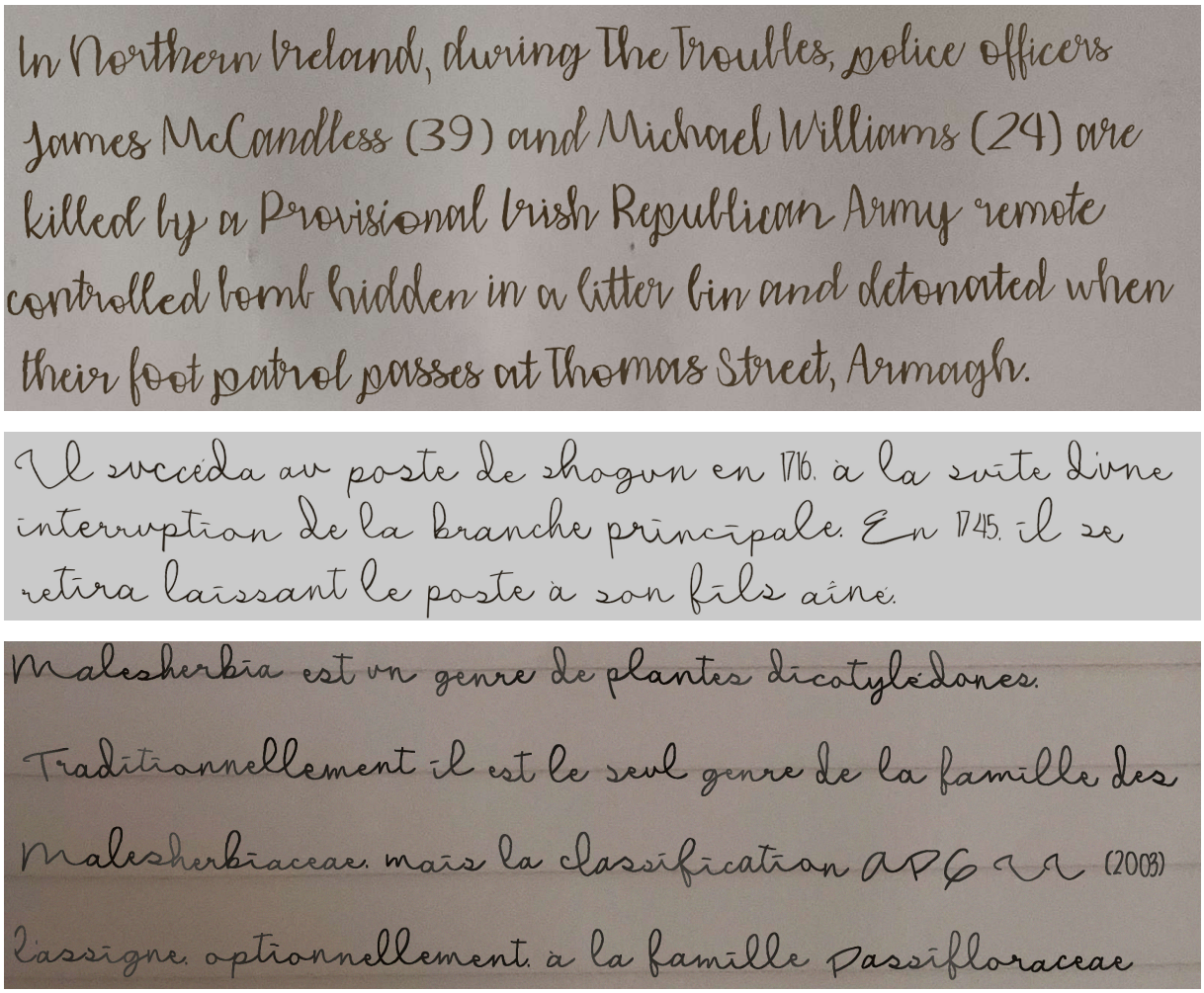


FIGURE A.8 – Exemples de paragraphes modernes qu'il est possible de générer avec notre approche.

COLLEGE DOCTORAL BRETAGNE
MATHS, TELECOMS
INFORMATIQUE, SIGNAL
SYSTEMES, ELECTRONIQUE

INSA INSTITUT NATIONAL DES SCIENCES APPLIQUÉES RENNES

Titre : Architectures de transformer légères pour la reconnaissance de textes manuscrits anciens

Mot clés : Apprentissage avec peu de données ; Transformer ; Reconnaissance d'écriture manuscrite ; Génération de données synthétiques ; Réseaux de neurones ; Architectures légères

Résumé : En reconnaissance d'écriture manuscrite, les architectures Transformer permettent de faibles taux d'erreur, mais sont difficiles à entraîner avec le peu de données annotées disponibles. Dans ce manuscrit, nous proposons des architectures Transformer légères adaptées aux données limitées. Nous introduisons une architecture rapide basée sur un encodeur Transformer, et traitant jusqu'à 60 pages par seconde. Nous proposons aussi des architectures utilisant un décodeur Transformer pour inclure l'apprentissage de la langue dans la reconnaissance des caractères. Pour entraîner efficacement nos architectures, nous proposons des algorithmes de génération de données synthétiques adaptées au style visuel des documents modernes et anciens. Nous proposons également des stratégies pour l'apprentissage avec peu de données spécifiques, et la réduction des erreurs de prédiction. Nos architectures, combinées à l'utilisation de données synthétiques et de ces stratégies, atteignent des taux d'erreur compétitifs sur des lignes de texte de documents modernes. Sur des documents anciens, elles parviennent à s'entraîner avec des nombres limités de données annotées, et surpassent les approches de l'état de l'art. En particulier, 500 lignes annotées sont suffisantes pour obtenir des taux d'erreur caractères proches de 5%.

Title: Lightweight Transformer-based Architectures for Historical Handwritten Text Recognition

Keywords: Learning with Limited Data; Transformer; Handwritten Text Recognition; Synthetic data generation; Neural networks; Lightweight architectures

Abstract: Transformer architectures deliver low error rates but are challenging to train due to limited annotated data in handwritten text recognition. We propose lightweight Transformer architectures to adapt to the limited amounts of annotated handwritten text available. We introduce a fast Transformer architecture with an encoder, processing up to 60 pages per second. We also present architectures using a Transformer decoder to incorporate language modeling into character recognition. To effectively train our architectures, we offer algorithms for generating synthetic data adapted to the visual style of modern and historical documents. Finally, we propose strategies for learning with limited data and reducing prediction errors. Our architectures, combined with synthetic data and these strategies, achieve competitive error rates on lines of text from modern documents. For historical documents, they train effectively with minimal annotated data, surpassing state-of-the-art approaches. Remarkably, just 500 annotated lines are sufficient for character error rates close to 5%.