*T*owards *E*nd-t*o*-end *H*andwritten D*o*cument Rec*o*gniti*o*n

Denis Coquenet

Normandie Université

THÈSE

Pour obtenir le diplôme de doctorat

Spécialité INFORMATIQUE

Préparée au sein de l'Université de Rouen Normandie

Towards End-to-end Handwritten Document Recognition

Présentée et soutenue par
DENIS COQUENET

Thèse soutenue le 29/09/2022 devant le jury composé de		
M. MATHIEU AUBRY	MAITRE DE CONFERENCES HDR, Ecole des Ponts ParisTech	Rapporteur du jury
M. CHRISTIAN WOLF	MAITRE DE CONFERENCES HDR, INSTITUT NATIONAL DES SCIENCES APPLIQUEES DE LYON	Rapporteur du jury
M. CLEMENT CHATELAIN	MAITRE DE CONFERENCES HDR, Institut National des Sciences Appliquées Rouen Normandie	Membre du jury
MME ELISA FROMONT	PROFESSEUR DES UNIVERSITES, UNIVERSITE RENNES 1	Membre du jury
M. HAROLD MOUCHÈRE	PROFESSEUR DES UNIVERSITES, UNIVERSITE NANTES	Membre du jury
M. THIERRY PAQUET	PROFESSEUR DES UNIVERSITES, Université de Rouen Normandie	Directeur de thèse

Thèse dirigée par THIERRY PAQUET (Laboratoire d'Informatique, du Traitement de l'Information et des Systèmes)

UNIVERSITÉ DE ROUEN NORMANDIE

MIIS

litis

Acknowledgments

First of all, I would like to thank my thesis supervisor Thierry Paquet and my advisor Clément Chatelain for their confidence, their support and all the advice and knowledge they have shared with me. I am grateful to Nicolas Malandain for this conversation which led me to consider undertaking a thesis. Thanks also to Yann Soullard who supervised and advised me during my first research works.

I would like to thank Christian Wolf and Mathieu Aubry who kindly accepted to review my thesis, and more generally to the whole defense committee: Harold Mouchere and Elisa Fromont. I am very honored and humbled by their review of my work.

I would like to thank the whole LITIS, for its welcome and its kindness, and more specifically the Apprentissage team. I would like to particularly thank Mélodie Boillet, with whom I went through the different stages of this thesis, for the laughs, the stimulating discussions and the "adventure moments" we shared together. I would like to thank Guillaume Renton, Thomas Constum and Theo Larcher without whom lunch breaks would not have been as pleasant. I would also like to thank all the administrative staff, who have accompanied me and whose daily work is remarkable: Fabienne Bocquet, Mathieu Baum and Brigitte Diarra, just to name a few.

I would also like to thank all the people who have been by my side during these three years of thesis. I would like to thank my family who has always supported me and believed in me. I thank my friends for the great moments shared with them, far from the worries of the thesis. I would like to take this opportunity to thank my love who has given me invaluable support.

I would like to thank the staff of CRIANN (Regional HPC Center, Normandy, France), notably Benoist Gaston and Beatrice Charton, and of GENCI-IDRIS (Grant 2020-AD011012155). This thesis was financially supported by the French Defense Innovation Agency and by the Normandy region, which I also thank.

Résumé

La reconnaissance de textes manuscrits a été largement étudiée au cours des dernières décennies pour ses nombreuses applications. Aujourd'hui, l'approche à l'état de l'art repose sur un processus en trois étapes. Le document est segmenté en lignes de texte, qui sont ensuite ordonnées et reconnues. Cependant, cette approche en trois étapes présente de nombreux inconvénients. Les trois étapes sont traitées indépendamment alors qu'elles sont étroitement liées. Les erreurs s'accumulent d'une étape à l'autre. L'étape d'ordonnancement est basée sur des règles heuristiques qui empêchent son utilisation pour des documents à la mise en page complexe ou pour des documents hétérogènes. L'étape de segmentation nécessite ses propres annotations supplémentaires.

Dans cette thèse, nous proposons de résoudre ces problèmes en effectuant la reconnaissance du texte du document en une seule étape, de bout en bout. Pour ce faire, nous augmentons progressivement la difficulté de la tâche de reconnaissance, en passant de lignes isolées à des paragraphes, puis à des documents entiers. Nous avons proposé une approche au niveau ligne, basée sur un réseau entièrement convolutif, afin de concevoir un premier module générique d'extraction de caractéristiques pour la tâche de reconnaissance d'écritures manuscrites.

Sur la base de ce travail préliminaire, nous avons étudié deux approches différentes pour reconnaître des paragraphes manuscrits. D'une part, nous avons conçu une approche non récurrente qui vise à aligner les prédictions du paragraphe entier dans un espace 2D afin de préserver la nature de l'image d'entrée. D'autre part, nous avons conçu une approche récurrente dans laquelle les lignes de texte sont traitées de manière itérative. Nous avons obtenu des résultats à l'état de l'art au niveau paragraphe sur les jeux de données RIMES 2011, IAM et READ 2016 et nous avons également dépassé l'état de l'art au niveau ligne sur ces jeux de données.

Nous avons enfin proposé la première approche de bout en bout dédiée à la reconnaissance à la fois du texte et de la mise en page, au niveau document. Les caractères et les symboles représentant la mise en page sont prédits séquentiellement en suivant un ordre de lecture appris. Nous avons proposé deux nouvelles métriques que nous avons utilisées pour évaluer cette tâche sur les jeux de données RIMES 2009 et READ 2016, au niveau page et double page.

Dans une optique de reproductibilité, de transparence et de partage scientifique auquel nous sommes attachés, tous les codes et poids des modèles sont publiquement disponibles en ligne.

Mots-clés **-** Reconnaissance de textes manuscrits, mécanismes d'attention, réseaux de neurones entièrement convolutifs, Transformer.

Abstract

Handwritten text recognition has been widely studied in the last decades for its numerous applications. Nowadays, the state-of-the-art approach consists in a three-step process. The document is segmented into text lines, which are then ordered and recognized. However, this three-step approach has many drawbacks. The three steps are treated independently whereas they are closely related. Errors accumulate from one step to the other. The ordering step is based on heuristic rules which prevent its use for documents with a complex layouts or for heterogeneous documents. The need for additional physical segmentation annotations for training the segmentation stage is inherent to this approach.

In this thesis, we propose to tackle these issues by performing the handwritten text recognition of whole document in an end-to-end way. To this aim, we gradually increase the difficulty of the recognition task, moving from isolated lines to paragraphs, and then to whole documents. We proposed an approach at the line level, based on a fully convolutional network, in order to design a first generic feature extraction step for the handwriting recognition task.

Based on this preliminary work, we studied two different approaches to recognize handwritten paragraphs. First, we designed a one-shot approach which aims at aligning the predictions of the whole paragraph in a two-dimensional space to preserve the nature of the the input image. Second, we designed a recurrent approach in which text lines are processed iteratively. We reached state-of-the-art results at paragraph level on the RIMES 2011, IAM and READ 2016 datasets and outperformed the line-level state of the art on these datasets.

We finally proposed the first end-to-end approach dedicated to the recognition of both text and layout, at document level. Characters and layout tokens are sequentially predicted following a learned reading order. We proposed two new metrics we used to evaluate this task on the RIMES 2009 and READ 2016 dataset, at page level and double-page level.

For reproducibility, transparency and scientific sharing purposes to which we are attached, all codes and weights of the models are publicly available online.

Keywords **-** Handwritten Text Recognition, Attention mechanism, Image-to-sequence, Fully Convolutional Network, Transformer.

Table of Contents

List of Tables

List of Figures

List of Algorithms

Acronyms

ANN Artificial Neural Network.

BLSTM Bidirectional Long-Short Term Memory.

CE Cross Entropy.

CER Character Error Rate.

CNN Convolutional Neural Network.

CTC Connectionist Temporal Classification.

CV Computer Vision.

DAN Document Attention Network.

DL Deep Learning.

DLA Document Layout Analysis.

DNN Deep Neural Network.

DPI Dots Per Inch.

DSC Depthwise Separable Convolution.

FCN Fully Convolutional Network.

FFN Feed-Forward Network.

GCNN Gated Convolutional Neural Network.

GCRL Gated Convolutional Recurrent Network.

GED Graph Edit Distance.

GFCN Gated Fully Convolutional Network.

GPU Graphic Processing Unit.

GRU Gated Recurrent Unit.

HDR Handwritten Document Recognition.

HMM Hidden Markov Model.

HTR Handwritten Text Recognition.

IoU Intersection over Union.

LM Language Model.

LOER Layout Ordering Error Rate.

LSTM Long-Short Term Memory.

mAP mean Average Precision.

MDLSTM Multi-Dimensional Long-Short Term Memory.

MDRNN Multi-Dimensional Recurrent Neural Network.

ML Machine Learning.

MLP Multi-Layer Perceptron.

NER Named Entity Recognition.

NLP Natural Language Processing.

NMT Neural Machine Translation.

OCR Optical Character Recognition.

PPER Post Processing Edition Rate.

RNN Recurrent Neural Network.

RPN Region Proposal Network.

seq2seq Sequence-to-sequence.

SGD Stochastic Gradient Descent.

SPAN Simple Predict & Align Network.

VAN Vertical Attention Network.

ViT Vision Transformer.

WER Word Error Rate.

XML eXtensible Markup Language.

Chapter 1

Introduction

1.1 Context and motivation

One of the inherent characteristics of human beings is their ability to communicate, to exchange information with each other. Since the birth of writing, a few thousand years ago, human beings used various media to store, save and distribute information: from gravings and papyri for the oldest to whiteboards, road signs, and mostly paper sheets nowadays. We are now living in a digital era. However, most of the information is only stored on physical media. Indeed, handwriting has long been ubiquitous in our daily lives no matter what form it takes: shopping lists, student notes, letters or even books.

Extracting the textual content from these media can be particularly useful for the automation of many industrial tasks such as bank check, invoice and form analysis and processing for example. It is also useful for historians needs, to avoid experts to transcribe the texts themselves, which is very time-consuming. Handwritten Text Recognition (HTR) can be used as a first step among more complex systems to translate documents or spot areas of interest through keywords. It also has interesting military applications: real-time document translation for spying for instance. This has induced a great interest in the automation of the handwriting recognition task.

HTR consists in recognizing the handwritten text from a digitized document, by producing a sequence of characters in ASCII format. More specifically, this thesis is dedicated to offline HTR, *i. e.* that the input is an image, a digital representation obtained after a scanning process. HTR is a difficult task which remains challenging even for modern computer vision systems. As a matter of fact, there is a large diversity of writing styles from one person to another: block letters, cursive, more or less inclined handwritings. In addition, the size of the characters, the thickness of the strokes and even the inter-character or inter-word spacing can vary significantly. The text can also be colored, with various backgrounds. All this variability makes this task more difficult than the recognition of printed text.

Nowadays, state-of-the-art approaches for the recognition of handwritten documents in the literature rely on deep neural networks. This task is mainly handled through a complex pipeline implying three main steps that are sequentially applied: the segmentation of the document into text regions (and sometimes other regions of interest such are drawings or tables), the ordering of these regions, and the recognition of the text regions. However,

this standard approach has several drawbacks. By definition, HTR only corresponds to the recognition of the text. The use of this three-step paradigm (segmentation, ordering, recognition) involves the training of a segmentation model which requires its own segmentation annotations, which are costly to produce. In addition, the ordering step is usually based on heuristic rules. Designing a rule-based ordering step is tedious for complex documents and is impossible without prior knowledge about the input documents, whereas the reading order is crucial to process whole documents. Moreover, the recognition errors accumulate with the errors made by the two previous stages.

Breaking down the HTR task into three steps prevents a global understanding of the document. Indeed, recognizing a whole document implies to read the different textual regions in a specific order to preserve the global coherence of the content; this is only feasible with an understanding of the layout. The layout may be very different from one document to another: one-column for this thesis, two-column for some scientific papers, multi-column for journals or more complex for maps and schemes. Many non-textual elements can also be found in the image, such as illustrations or mathematical expressions. HTR is a more complicated task than it seems: it implies the localization and the recognition of the different characters in the correct order, ignoring the non-textual parts, and based on the understanding of the document layout.

In addition, managing the three steps altogether is difficult: the different stages being interdependent, the modification of one of the steps may imply the adaptation of the whole pipeline. This way, the work proposed in the literature mainly focused on only one of these steps. The recognition of isolated character, word and line images, and their corresponding segmentation stage, have been widely studied by the community, leading to state-of-the-art performances. To alleviate the need for segmentation labels, very few works have been devoted to the end-to-end recognition of paragraphs. Although essential, the reading order aspect is almost non-existent in the literature.

This way, processing the HTR task in an end-to-end way, *i. e.* designing a model which takes advantage of the layout analysis and which integrates the notion of reading order to recognize the text, seems to be to consider to solve these different issues and limitations.

1.2 Contributions

This thesis aims at advancing the field of handwriting recognition through four contributions which are articulated around a common goal: to move towards the end-to-end recognition of whole documents. To this end, we started from the recognition of isolated text lines to go towards the recognition of paragraphs, then of whole documents.

Contribution for line-level HTR. We started with the idea of designing a generic feature extractor module (encoder) for the HTR task, which could be used for text line recognition, but also for paragraph and document recognition. To this end, we proposed a Fully Convolutional Network (FCN) architecture we applied on the HTR task at line level. We achieved competitive results on two public datasets: RIMES 2011 [1] and IAM [2].

Contributions for paragraph-level HTR. Based on the previous contribution, we then increased the level of reading order difficulty by dealing with paragraph images: this represents an additional vertical axis to take into account. We proposed two approaches for the HTR task at paragraph level:

- A one-shot approach *i. e.* the prediction of the whole paragraph is carried out in a single iteration, without any recurrence. The idea is to preserve the two-dimensional nature of the task for the prediction. A reshaping operation is then used to switch from a two-dimensional prediction to a one-dimensional character sequence. The order of the characters is preserved through a vertical alignment during the prediction. We proposed the Simple Predict & Align Network (SPAN) following this approach. This recurrence-free model reached competitive results on three public datasets: RIMES 2011, IAM and READ 2016 [3].

- A line-level recurrent approach *i. e.* the paragraph image is processed iteratively, each iteration being dedicated to the recognition of a specific text line. We proposed the Vertical Attention Network (VAN) following this approach through the use of a line-level hybrid attention module. The VAN also learns to detect the end of the paragraph so as to stop the recurrent process. It outperformed state-of-the-art results on the three public datasets: RIMES 2011, IAM and READ 2016.

Contribution for Handwritten Document Recognition (HDR). The last step was to handle whole documents. It means handling images containing multiple paragraphs, sometimes spread over several columns. We proposed the first approach able to deal with whole document images in an end-to-end fashion. In addition to the text recognition, the proposed approach aims at recognizing the layout by labeling text parts using begin and end tags in an XML-like fashion. We proposed the Document Attention Network (DAN) to illustrates this approach. It consists in a segmentation-free transformer-based model which involves a character-level attention. This way, the DAN is trained without using any physical segmentation annotation. We proposed two new metrics for this task and we evaluated the model at single-page and double-page levels on the RIMES 2009 [4] and READ 2016 datasets.

1.3 Overview of the thesis

This manuscript is divided into 6 chapters. Chapter 2 formally presents the HTR task as an image-to-sequence problem. The general deep learning background is provided, with a particular focus on the areas of Computer Vision (CV) and Sequence-to-sequence (seq2seq) transformation problems. The three following chapters are guided by the evolution of the approaches to deal with the HTR task and our contributions. Chapter 3 is dedicated to line-level HTR approaches. The paragraph-level HTR approaches are studied in Chapter 4. We present the document-level approach we proposed in Chapter 5. Conclusion and future directions of work are given in Chapter 6.

Chapter 2

Handwritten text recognition: overview of the problem

The task of Handwritten Text Recognition (HTR) can be formulated as follows. The aim is to recognize all the text from an input image. This image is noted $\boldsymbol{X} \in \mathbb{R}^{H \times W \times C}$, where H is its height, W is its width and C is the number of channels (1 for gray-scaled images and 3 for RGB images). The image contains a sequence $\boldsymbol{y}$ of L characters $\boldsymbol{y^i}$ from an alphabet $\mathcal{A}$. This sequence of characters is structured in multiple text regions (or paragraphs) according to a specific layout. The goal is to develop a system that takes as input the image $\boldsymbol{X}$ and outputs all the characters, in a human reading order, as the prediction $\hat{\boldsymbol{y}}$. For an English paragraph for example, it means from top to bottom and from left to right. The reading order can be more complicated for whole documents, from one paragraph to another for instance.

In other words, during training, the aim is to maximize $p(\hat{\boldsymbol{y}} = \boldsymbol{y}|\boldsymbol{X})$, the probability to predict the ground truth sequence $\boldsymbol{y}$, given the input image $\boldsymbol{X}$. One can implement the maximization of this quantity using different statistical models; in this thesis, we focus on optimization approaches associated with Deep Neural Network (DNN) architectures. The task to be solved is depicted in Figure 2.1. Here, the line breaks are considered as any other character, but it could also be considered as a space character.

One should note several points to understand the complexity of the task:

- We do not make any assumptions about the digitized handwritten documents in terms of writing styles, layout, resolution, background or color/format encoding.
- We do not make any assumptions about the size of the digitized document, so the input size can vary a lot from one image to another. Moreover, the image size is not related to the number of characters in the image *i. e.* the length of the output sequence. This length is not known beforehand.
- We do not make any assumptions about the reading order of the documents. For example, a single-column document follows a fixed reading order: from top to bottom and from left to right whereas a two-column document implies another reading-order level: column by column.

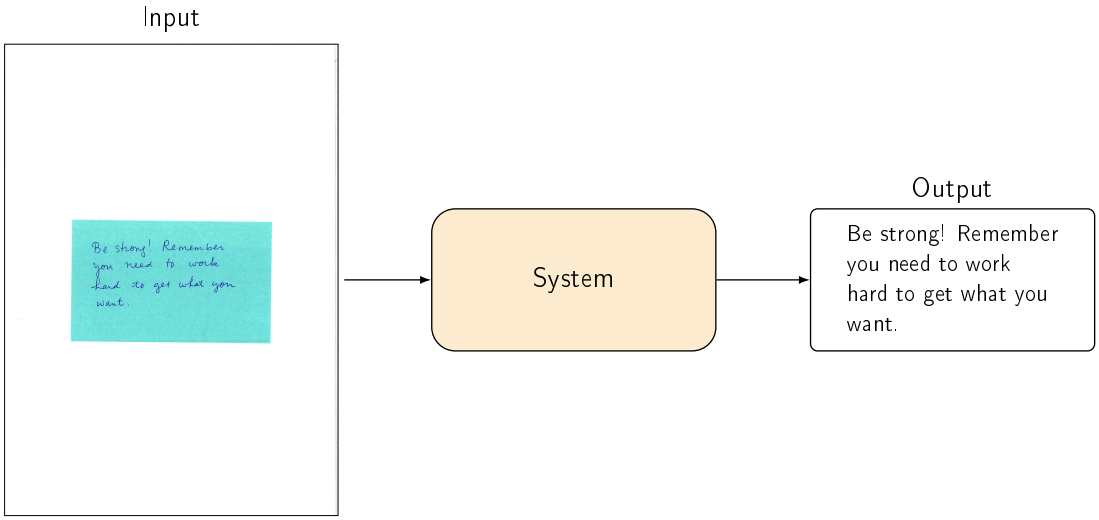


Figure 2.1: Handwritten text recognition task.

- For complex documents such as schemes or maps, many reading orders would be humanly acceptable: there is not a unique valid output sequence. This should be taken into account when training the system.

Nowadays, state-of-the-art results for such a complex task are obtained with deep neural networks. Instinctively, one could classify HTR as a Computer Vision (CV) task since it involves an input image. But it could also be seen as a sequence-to-sequence (seq2seq) problem due to the expected output character sequence. In fact, we will see that it would be better to qualify HTR as an image-to-sequence problem.

In the following sections, we explain how a DNN system works and present the main challenges when dealing with such systems. We go over the main discoveries and improvements in the fields of computer vision and sequence-to-sequence problems, which will give us the fundamental elements to handle image-to-sequence tasks. The remainder of this chapter is dedicated to the presentation of the public datasets and metrics we used to evaluate the performance of the trained networks.

2.1 Deep Learning background

Deep Learning (DL) is part of Machine Learning (ML): it aims at estimating a function $F : \boldsymbol{x} \rightarrow \hat{\boldsymbol{y}}$ in order to solve a given task, be it regression or classification, so as to optimize the objective $p(\hat{\boldsymbol{y}} = \boldsymbol{y}|\boldsymbol{x})$. While older ML approaches rely on a handcrafted feature extraction process, deep learning techniques are based on learning this feature extraction process, through the use of Artificial Neural Networks (ANNs), applied on raw data. The ANNs used in DL differ from the other ML approaches by the way they combine many elementary functions to estimate a complex function F. Indeed, ANNs are inspired by the biological neural networks found in living organisms: they are made of a large quantity of neurons, connected to each other.

Perceptron

In 1943, the authors of [5] described the first artificial neuron. Few years later, in 1958, Frank Rosenblatt proposed an algorithm to adjust the weights, also adding few changes to this first neuron, leading to the Perceptron. The Perceptron is a binary classifier based on a unique formal neuron followed by a heaviside function $\mathcal{H}$. It is depicted in Figure 2.2 for a 2-feature input.

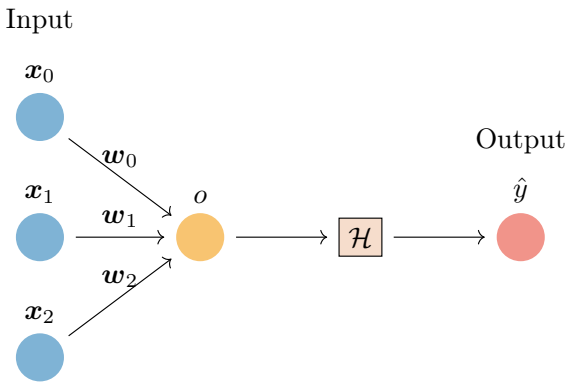


Figure 2.2: Perceptron.

The formal neuron is fully connected to the inputs $\boldsymbol{x} = [\boldsymbol{x}_1, ..., \boldsymbol{x}_n]$ *i. e.* its output o is a weighted sum of its inputs with its trainable weights $\boldsymbol{w}_i$:

$$o = \sum_{i=1}^{n} \boldsymbol{w}_i \boldsymbol{x}_i + b, \tag{2.1}$$

where b is known as bias. It is also a trainable parameter. By setting $x_0 = 1$ and $w_0 = b$, one can re-write the equation:

$$o = \sum_{i=0}^{n} \boldsymbol{w}_i \boldsymbol{x}_i. \tag{2.2}$$

The output of the Perceptron $\hat{y}$ is then computed by applying the heaviside function $\mathcal{H}$ on o:

$$\hat{y} = \mathcal{H}(o) = \begin{cases} 0 & \text{if } o < 0, \\ 1 & \text{otherwise.} \end{cases} \tag{2.3}$$

One can note that the decision function of the Perceptron is linear, which is an important limitation for real-world classification tasks.

Multi-Layer Perceptron

The Multi-Layer Perceptron (MLP) was designed to tackle the linear limitation of the Perceptron. It is made up of a succession of neuronal layers, which are arranged one after the other; connections are only made between adjacent layers, and always from the input to the output, thus the name Feed-Forward Network (FFN). More precisely, in an MLP, the layers

are fully-connected; it means that each neuron of a layer is connected to all the neurons of the previous layer. An example of a MLP with 3 hidden layers is represented in Figure 2.3. Activation functions are used at the output of each layer. The introduction of non-linear activation functions enables to overcome the major drawback of the original Perceptron. MLP was the most popular and simple neural network architecture before the advent of deep learning. It was common to only use a single hidden layer. Nowadays, deep learning mainly refers to the use of neural networks made up of several neuronal layers. The number of neuronal layers corresponds to the depth of the network.

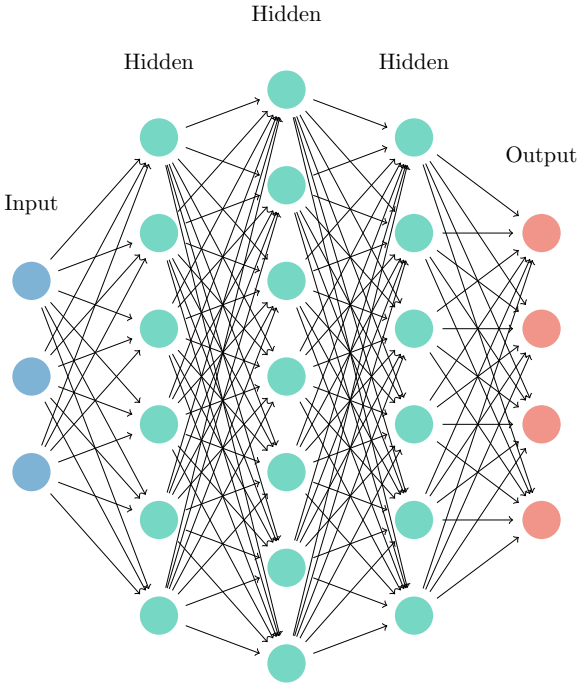


Figure 2.3: Multi-Layer Perceptron.

More generally, an FFN can be represented as a stack of γ neuronal layers. Such a system is depicted in Figure 2.4. Each neuronal layer $\boldsymbol{H}_i \in \mathbb{R}^{n_i}$ is the result of an operation performed on the neurons of the previous layer $\boldsymbol{H}_{i-1}$. This operation can imply some variables, known as trainable weights (or trainable parameters). These weights can be represented as a matrix $\boldsymbol{W}_i \in \mathbb{R}^{n_i \times n_{i-1}}$. But it can also be a parameter-free operation, such as an average computation. The aggregation of all these elementary functions enables to model a complex function. To make the notations simpler, we denote $\boldsymbol{\theta}$ the vector composed of all the weights from all the layers $(\{\boldsymbol{W}_1, ..., \boldsymbol{W}_\gamma\})$.

Training

Training a neural network consists in finding the optimal weights $\boldsymbol{\theta}^*$ that minimize the errors made by the model. Deep learning approaches rely on gradient-based algorithms, as many other machine learning approaches. Indeed, DL approaches use gradient backpropagation algorithms [6], known as optimizers, in order to iteratively improve the weights through

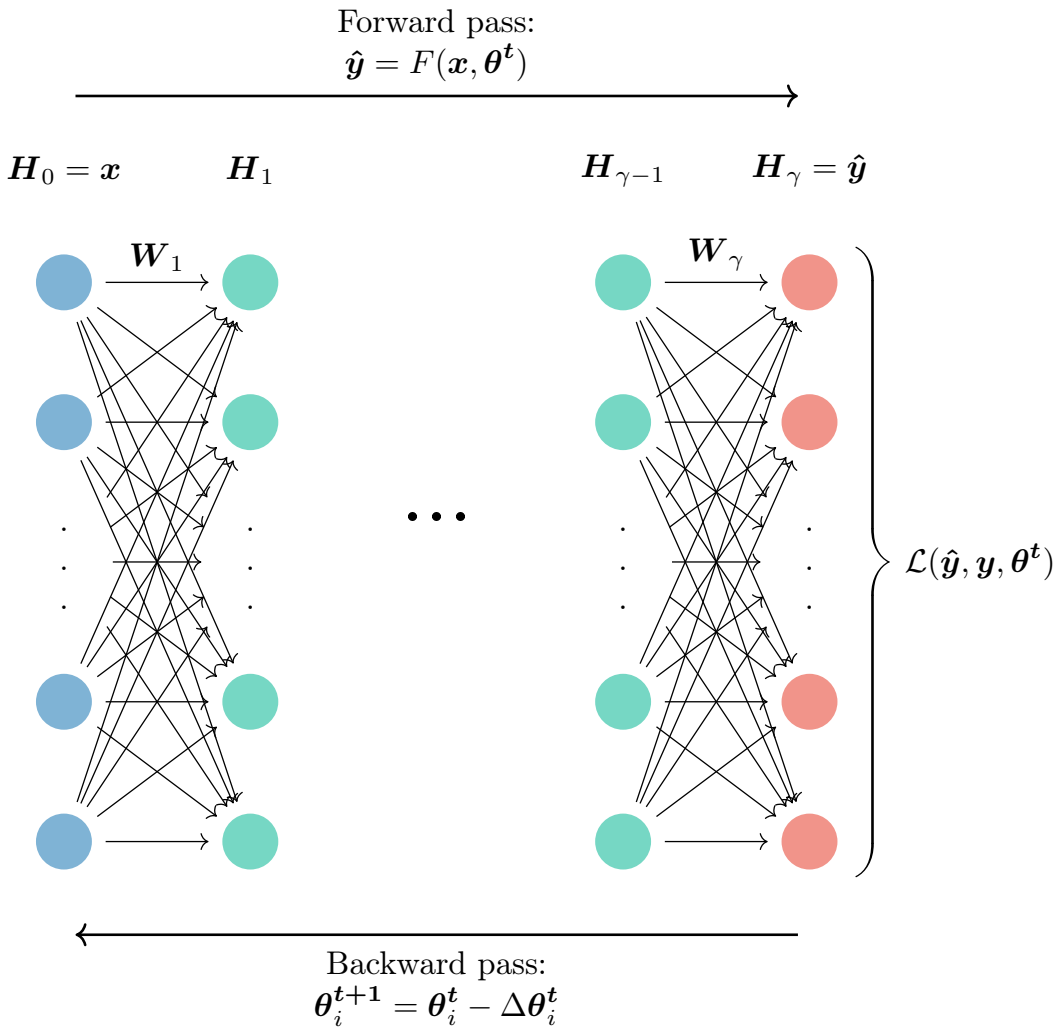


Figure 2.4: Neural network overview.

examples *i. e.* to modify the values of each weight in such a way that the system improves its predictions. This falls into the supervised learning paradigm.

It implies the use of a dataset $\mathcal{D}_{\text{train}}$ composed of couples $(\boldsymbol{x}, \boldsymbol{y})$, where $\boldsymbol{x}$ are the inputs and $\boldsymbol{y}$ are their respective expected output (known as the ground truth). One must then define a loss function $\mathcal{L}$. It must be a differentiable function with respect to the parameters $\boldsymbol{\theta}$, measuring the prediction error for a given sample *i. e.* the distance between the prediction of the model $\hat{\boldsymbol{y}}$ and the expected output $\boldsymbol{y}$. Mean Square Error (MSE) and Cross Entropy (CE) are losses that are commonly used for regression and classification tasks, respectively. The average of the loss functions over the whole training dataset is usually taken as the objective function to minimize. We refer to it as the cost function $\mathcal{C}$. To sum up, the goal is to find:

$$\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} \mathcal{C}(\mathcal{D}_{\text{train}}, \boldsymbol{\theta}) = \arg\min_{\boldsymbol{\theta}} \sum_{(\boldsymbol{x},\boldsymbol{y})\in\mathcal{D}_{\text{train}}} \mathcal{L}(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{\theta}), \tag{2.4}$$

which is solved by gradient descent.

The gradient descent algorithm consists in iteratively updating the weights $\boldsymbol{\theta}_i$ so as to decrease the loss function. $\boldsymbol{\theta}_i^{\boldsymbol{t}}$ denotes the weight i at iteration t. By iteration, we mean the process of updating the weights. The weights update consists in increasing the weights $\boldsymbol{\theta}_i^{\boldsymbol{t}}$ by a fraction of the opposite of the gradient vector of the loss function $\mathcal{L}$ with respect to $\boldsymbol{\theta}_i^{\boldsymbol{t}}$, so as to get closer to the minimum of $\mathcal{L}$. This gradient vector can be computed on a single example or averaged over many examples, known as a batch:

$$\boldsymbol{\theta}_i^{\boldsymbol{t+1}} = \boldsymbol{\theta}_i^{\boldsymbol{t}} - \Delta\boldsymbol{\theta}_i^{\boldsymbol{t}}, \tag{2.5}$$

with

$$\Delta\boldsymbol{\theta}_i^{\boldsymbol{t}} = \eta \cdot \frac{1}{\text{card}(\mathcal{X}_{\text{batch}})} \cdot \sum_{(\boldsymbol{x},\boldsymbol{y})\in(\mathcal{X}_{\text{batch}},\mathcal{Y}_{\text{batch}})} \frac{\delta\mathcal{L}(\boldsymbol{x},\boldsymbol{y},\boldsymbol{\theta}^{\boldsymbol{t}})}{\delta\boldsymbol{\theta}_i^{\boldsymbol{t}}}, \tag{2.6}$$

where η is a fixed learning rate. This is known as the backward pass.

To compute this gradient vector, one must first compute the prediction $\hat{\boldsymbol{y}}$ based on the input $\boldsymbol{x}$ and on the current weights $\boldsymbol{\theta}^{\boldsymbol{t}}$, this is the forward pass:

$$\hat{y} = F(\boldsymbol{x}, \boldsymbol{\theta}^{\boldsymbol{t}}). \tag{2.7}$$

In addition, this weight update must be computed for each single weight of the neural network, no matter the neural layer. This is made possible by computing the gradient layer by layer, using the chain rule between two successive layers:

$$\frac{\delta\mathcal{L}}{\delta\boldsymbol{\theta}_i} = \frac{\delta\mathcal{L}}{\delta\boldsymbol{H}_i} \cdot \frac{\delta\boldsymbol{H}_i}{\delta\boldsymbol{\theta_i}}, \tag{2.8}$$

with

$$\frac{\delta\mathcal{L}}{\delta\boldsymbol{H}_i} = \frac{\delta\mathcal{L}}{\delta\boldsymbol{H}_{i+1}} \cdot \frac{\delta\boldsymbol{H}_{i+1}}{\delta\boldsymbol{H}_i}, \tag{2.9}$$

thus the name error gradient backpropagation, or backward pass.

Algorithm 1 sums up these different steps by presenting the standard optimizer to solve this problem. Weights are randomly initialized. The algorithm consists in going through the whole training dataset, again and again, successively updating the weights. These loops are known as epochs. The process stops when a stopping condition is met. Generally, it is a threshold on the variation of the value of a loss or metric through the last epochs; but it can also be a fixed number of epochs or training time. Between each epoch, training samples are shuffled to avoid bias during training. The update of the weights is carried out after processing N examples; N is called the mini-batch size. The choice of N is discussed in section 2.1.2.1. Processing a labeled example $(\boldsymbol{x}, \boldsymbol{y})$ consists in two steps: the forward pass and the backward pass, as defined previously.

This iterative optimization algorithm seems relatively simple. However, training a deep neural network faces three main difficulties:

- Ensuring generalization. The network is trained to minimize the cost function on a training dataset. The model must learn enough from these annotated examples to perform well on unseen data. However, it should not learn too much in order to avoid learning the training data exactly, without capacity to generalize on unseen data.

Algorithm 1: Standard error gradient backpropagation optimizer.

input: F, initialized with random weights $\boldsymbol{\theta}_i^{\mathbf{0}}$,
Learning rate η,
Dataset $\mathcal{D}_{\text{train}} = (\mathcal{X}_{\text{train}}, \mathcal{Y}_{\text{train}})$

$t \leftarrow 0$
while *stopping criterion not reached* **do**
  $(\mathcal{X}_{\text{shuffled}}, \mathcal{Y}_{\text{shuffled}}) \leftarrow \text{shuffle}(\mathcal{X}_{\text{train}}, \mathcal{Y}_{\text{train}})$
  for $(\mathcal{X}_{\text{batch}}, \mathcal{Y}_{\text{batch}}) \in \text{split}(\mathcal{X}_{\text{shuffled}}, \mathcal{Y}_{\text{shuffled}})$ **do**
    $\Delta\boldsymbol{\theta}_i^{\boldsymbol{t}} \leftarrow 0 \; \forall i$
    for $(\boldsymbol{x}, \boldsymbol{y}) \in (\mathcal{X}_{\text{batch}}, \mathcal{Y}_{\text{batch}})$ **do**
      $\hat{\boldsymbol{y}} \leftarrow F(\boldsymbol{x}, \boldsymbol{\theta}^{\boldsymbol{t}})$
      $\Delta\boldsymbol{\theta}_i^{\boldsymbol{t}} \leftarrow \Delta\boldsymbol{\theta}_i^{\boldsymbol{t}} + \frac{\delta\mathcal{L}(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{\theta}^{\boldsymbol{t}})}{\delta\boldsymbol{\theta}_i^{\boldsymbol{t}}} \; \forall i$
    $\Delta\boldsymbol{\theta}_i^{\boldsymbol{t}} \leftarrow \eta \frac{\Delta\boldsymbol{\theta}_i^{\boldsymbol{t}}}{\text{card}(\mathcal{X}_{\text{batch}})} \; \forall i$
    $t \leftarrow t + 1$
    $\boldsymbol{\theta}_i^{\boldsymbol{t}} \leftarrow \boldsymbol{\theta}_i^{\boldsymbol{t-1}} - \Delta\boldsymbol{\theta}_i^{\boldsymbol{t-1}} \; \forall i$

- Converging to the global minimum of $\mathcal{C}$ is the main objective, *i. e.* reaching the optimal weights $\boldsymbol{\theta}^*$. Real-world tasks involve non-convex optimizations, and reaching the global minimum is not guaranteed. This way, one tries to converge as efficiently as possible. It means converging towards a local minimum near the global minimum, as fast as possible.

- Dealing with vanishing and exploding gradients. Neural networks are getting deeper and deeper in order to approximate more complex functions. This leads to vanishing or exploding gradients for the lowest layers of the network, implying a stagnation of the weights or overflows, respectively.

In the following, we discuss the main techniques that have helped to overcome or mitigate these various challenges, leading to more stable, robust and efficient deep networks.

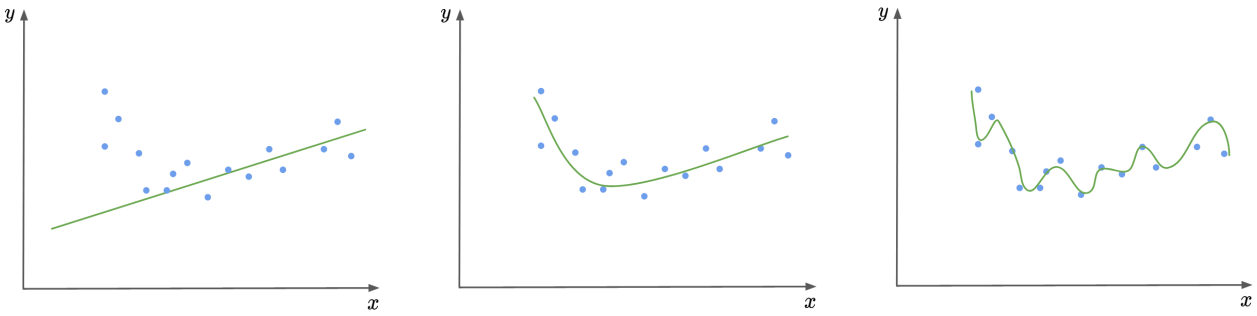
2.1.1 Generalization

So far, we have seen the global approach to minimize the cost function over the training dataset, through gradient descent. However, the real purpose of the network is to generalize on data never seen during training. Here comes the bias-variance dilemma [7]. It is defined as the conflict induced when trying to minimize these two sources of errors:

- The bias error. It corresponds to the error made from wrong assumptions of the model. A high bias is characterized by a poor modeling of the relationship between inputs and outputs. It is called underfitting.

- The variance error. It refers to errors induced by a sensitivity to small fluctuations in the training dataset. A high variance results in modeling noise from the training dataset, which is not relevant. This phenomenon is called overfitting.

These two cases are depicted in Figure 2.5, with a simple case of monovariate linear regression task. As one can note, the aim is to have a good generalization property *i. e.* a trade-off between bias and variance to adapt as best as possible to unseen data.



(a) Underfitting. (b) Generalization. (c) Overfitting.

Figure 2.5: Function approximation on the training dataset.

Choosing the network complexity *i. e.* the right number of parameters can be seen as the main issue. Indeed, the underfitting phenomenon depicted in Figure 2.5a can be explained by a too low number of trainable parameters (only two for a straight line). In contrast, too many parameters result in learning the training data too much, which leads to overfitting (Figure 2.5c). Generalization is obtained when choosing a consistent number of parameters with respect to the complexity of the task. However, this value cannot be known in advance and one generally chooses a number of parameters high enough to avoid underfitting. The overfitting issue is commonly handled by using regularization techniques and a specific criterion to stop the learning process.

Another key point regarding the generalization issues is the amount of available labeled data: the more training data, the less overfitting. This is all the more true as the number of parameters increases. However, the annotation of the data is mainly a hand-made process, which is costly. This way, it is crucial to deal with as few manually labeled data as possible.

In the following, we present the main techniques we mentioned to deal with the generalization issues.

2.1.1.1 Regularization

Neural networks are prone to overfitting *i. e.* the model learns features from the training set that are not relevant for the task, leading to poor generalization to unseen data. Regularization techniques aims at reducing this phenomenon.

A first regularization technique consists in adding a regularization term on the weights to the loss function $\mathcal{L}$. The idea is that one does not know in advance the complexity of the model one wants to approximate trough the neural network. By default, one chooses a model with a high complexity *i. e.* with a high number of parameters. However, the use of too many parameters amplifies the overfitting phenomenon. Intuitively, one would like to add a penalization on the number of parameters used in the model; but the 0-norm is not differentiable and cannot be used. A way to counteract this issue is to use the L1 and L2 regularizations [8], in order to introduce sparsity, with some weights tending to 0. This

way, it gets a similar impact as controlling the number of parameters of the network, while keeping a differentiable cost function.

L1 regularization applies 1-norm to the weights, it is defined as follows:

$$\mathcal{L}_{\text{L1}}(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{\theta}) = \mathcal{L}(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{\theta}) + \lambda \sum_j |\boldsymbol{\theta}_j|, \tag{2.10}$$

where λ is the regularization parameter. In the same way, L2 regularization applies 2-norm to the weights:

$$\mathcal{L}_{\text{L2}}(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{\theta}) = \mathcal{L}(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{\theta}) + \lambda \sum_j \boldsymbol{\theta}_j^2. \tag{2.11}$$

The introduction of sparsity aims at mimicking a network with fewer parameters. It leads to less complex models, avoiding them to overfit too much. However, L1 and L2 regularizations do not prevent overfitting totally. Moreover, they do not solve the co-adaptation issue.

Co-adaptation is the phenomenon whereby all neuronal connections do not carry the same amount of information. This leads to some connections being very prevalent and others insignificant in the decision. Dropout [9] was proposed in 2014 as a new regularization approach to tackle this issue. Dropout is a technique which is used at training time only. It consists in randomly zeroing some elements of a hidden neural layer, given a probability τ, using a Bernoulli distribution. To compensate the introduction of zeros, the others values are scaled by $\frac{1}{1-\tau}$ to preserve the behavior between training time and evaluation time. This way, Dropout pushes all neurons to contribute to the prediction since a part of them is randomly ignored at each training step, making the network more robust. Spatial Dropout (or 2D Dropout) [10] is a similar approach proposed for the case of CNN where standard Dropout is less efficient since convolutional kernels are applied on neighbors, which may be strongly correlated. With Spatial Dropout, entire feature maps are ignored to prevent the network from using these correlations.

Even using regularization techniques, the model still tends to overfit through the epochs, notably due an excessive over-parameterization. One way to tackle this issue is to stop the training process before learning too much the training samples.

2.1.1.2 Stopping criterion

The network is trained on a training dataset $\mathcal{D}_{\text{train}}$. But the aim is to have the best generalization over unseen data. To this end, the original dataset $\mathcal{D}$ is generally split into three sub-datasets: $\mathcal{D}_{\text{train}}$, $\mathcal{D}_{\text{validation}}$ and $\mathcal{D}_{\text{test}}$. The model weights are updated to minimize the prediction errors with respect to $\mathcal{D}_{\text{train}}$. During training, one regularly estimates the error on $\mathcal{D}_{\text{validation}}$, which contains samples unseen from the model during training. This enables to select the weights that best generalize on these unseen examples. As shown on Figure 2.6, one should select weights just before the validation curve goes back up. The validation curve to look at can also be a metric curve. Indeed, metrics are similar to losses in that they evaluate a distance between prediction and ground truth. But metrics do not need to be differentiable since they are not used in the backpropagation process. This way, they can be designed with more flexibility and give a better idea of the performance of the model.

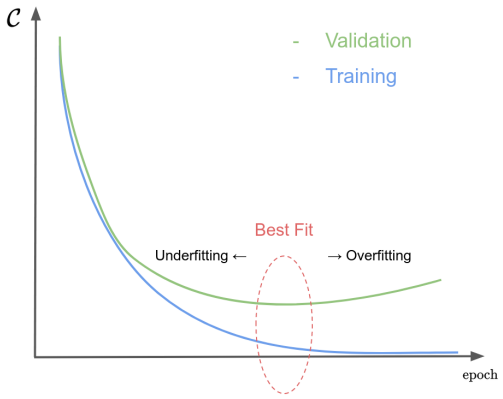


Figure 2.6: Stopping criterion.

However, this validation error estimation is not sufficient since it is biased. Indeed, we selected the weights for which the error was the lowest on these examples. In addition, hyperparameters are tuned to reach the lowest error on this sub-dataset. That is why a better estimation of the performance of the model on never-before-seen data is computed on $\mathcal{D}_{\text{test}}$.

Combining regularization techniques with this stopping criterion enables to alleviate the overfitting phenomenon. But it can be not sufficient if one does not have enough labeled data to train the model.

2.1.1.3 Dealing with few training data

A key point to reach a good convergence, avoiding overfitting and reaching great results, is the amount of available labeled data. There are several ways to deal with few labeled data.

Data augmentation

Data augmentation is a simple technique that enables to augment the number of training data artificially. It consists in applying transformations on the input data while preserving the relevant information *i. e.* the associated ground truth remains the same. The idea is to generate new training samples without additional cost. Moreover, it enables the model to determine which features are important and which are not since, for a given sample, the different data augmentation techniques generate different inputs while the expected outputs are the same. These data augmentation techniques are as varied as there are tasks to solve. For instance, color distortion, Gaussian noise addition, or rotation are common data augmentation techniques used for computer vision tasks. The data augmentation techniques we used for HTR are described in Appendix A.

Using synthetic data is another way to artificially augment the number of training data. Synthetic data are computer-generated examples based on heuristics or generative models. The aim is to produce correct couples $(\boldsymbol{x}, \boldsymbol{y})$ as realistic as possible for a given task without any human cost for annotation.

Transfer learning

Transfer learning is another way to avoid overfitting. Indeed, in many cases, one can face a lack of data for a target task T. Transfer learning is a way to reduce this issue by transferring knowledge from a network trained on a source task S correlated with T. This notion is illustrated in Figure 2.7.

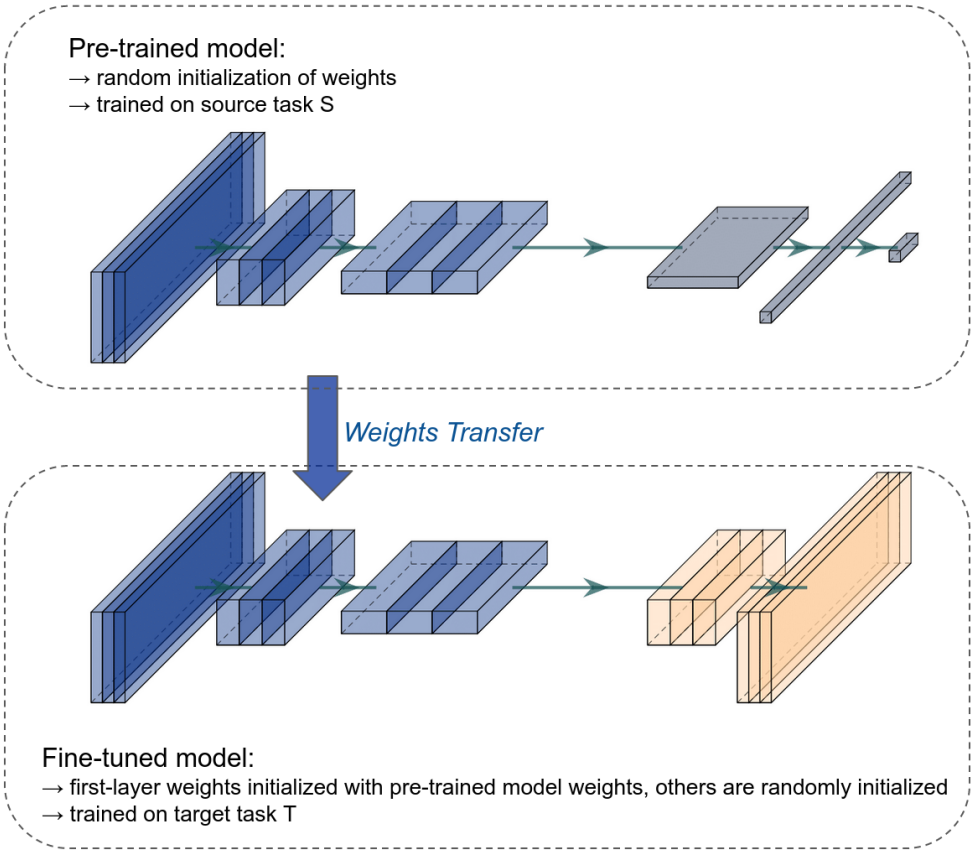


Figure 2.7: Transfer learning concept.

It consists in two steps:

- Pre-training: training a model on a source task S, whose weights are randomly initialized.
- Fine-tuning: training another model on a target task T, with a part of the weights initialized with the weight values of the pre-trained model.

It implies that both networks share, at least in part, the same architecture. It usually corresponds to the lowest layers of the network. The closer the tasks, the more effective this technique. As a matter of fact, transfer learning is inspired by human learning behavior. Someone who knows how to play a musical instrument will learn another instrument more easily than someone who has never played any instrument. On the other hand, the same observation can be made for learning a new score, for the same instrument. This way, transfer learning can also be applied for a same task, if we have a source dataset, different from the target one.

Self-supervised learning

Self-supervised learning is a specific kind of pre-training strategy which is notably used when there is no dataset available for the source task. Self-supervised learning consists in training part of the model from unlabeled data, still using a supervised learning strategy. The idea is to produce artificial labels (known as pseudo labels) from the input itself automatically *i. e.* without any human effort to generate the annotations. Those labels are then used for pre-training, this is also known as training on a pretext task. Auto-encoders [11] are an example of self-supervised learning. It consists in reconstructing the input signal. It means that the expected output (label) is the input. Another example is the completion of sequences of tokens whose some parts are masked, as in [12].

A recent trend among the self-supervised learning strategies is contrastive learning [13]. It is based on the idea of learning similarities and dissimilarities between examples. It generally consists in applying different data augmentation strategies on a same input. This training aims at learning to bring closer representations that come from the same input and move away those of examples coming from different inputs, or containing different information given a desired task.

Multi-tasking

Multi-tasking is close to transfer learning. Indeed, the aim is to benefit from learning of one or many other tasks. This induces the use of several models that share some of their weights. The main difference with standard transfer learning strategies is that the different models are trained jointly and not one after the other. The models can either be learned by changing the task between each weight update, or be learned together via a combination of the different losses if the input is the same for the different tasks. It also enables to reduce the total amount of trainable weights. This have been applied in many fields such as Natural Language Processing (NLP) [14], combining six tasks, namely Named Entity Recognition (NER), part-of-speech tagging, chunking, semantic role labeling, language modeling and semantically related words or CV with an object detection approach [15].

Semi-supervised learning

Semi-supervised learning [16] is another way to deal with small datasets. It is a special case where one has many unlabeled data (without human annotation), in addition to some annotated data. The aim is to take benefits from the use of the unlabeled data. One way to do this is to first train a model with the annotated data until reaching rather good results. Then, the model generates predictions for part of the unseen unlabeled data. If the predictions are judged correct enough according to a given criterion, they are used as pseudo labels, leading to new training samples. This process is repeated while the performance is increasing.

2.1.2 Converging efficiently

We have studied the general gradient descent algorithm used for training and we have seen how to handle the issues related to underfitting and overfitting. We now discuss the conver-

gence issues. Indeed, the aim is to converge *i. e.* to reach the global minimum of the cost function over the training dataset by updating the values of the weights. The main issue is to find how to modify the values of the weights in the right proportion, *i. e.* to go quickly towards the local minimum.

The Newton's method was proposed to tackle this problem. It consists in iteratively minimizing the derivative of the loss through its second-order Taylor approximation:

$$\mathcal{L}(\boldsymbol{\theta^t} + \epsilon) \approx \mathcal{L}(\boldsymbol{\theta^t}) + \nabla\mathcal{L}(\boldsymbol{\theta^t})\epsilon + \nabla^2\mathcal{L}(\boldsymbol{\theta^t})\epsilon^2 \tag{2.12}$$

with $\boldsymbol{\theta^{t+1}} = \boldsymbol{\theta^t} + \epsilon$.

To simplify the notations in the following, we consider a batch size of 1:

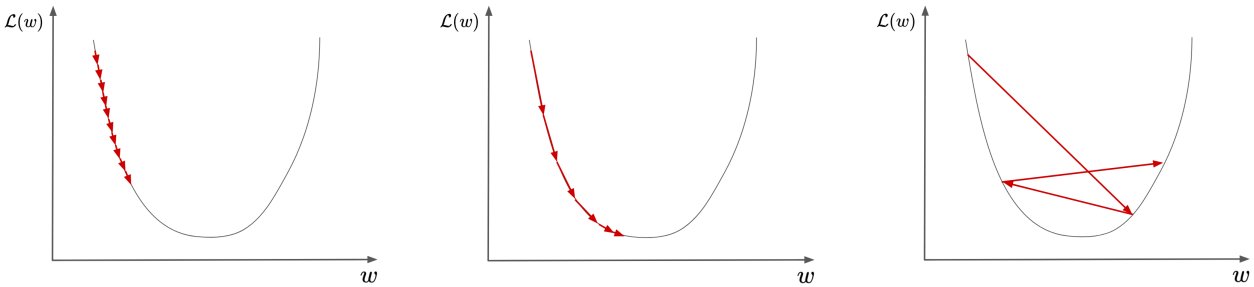
$$\boldsymbol{g}_i^t = \frac{1}{\text{card}(\mathcal{X}_{\text{batch}})} \cdot \sum_{(\boldsymbol{x},\boldsymbol{y})\in(\mathcal{X}_{\text{batch}},\mathcal{Y}_{\text{batch}})} \frac{\delta\mathcal{L}(\boldsymbol{x},\boldsymbol{y},\boldsymbol{\theta^t})}{\delta\boldsymbol{\theta}_i^t} = \frac{\delta\mathcal{L}}{\delta\boldsymbol{\theta}_i^t}. \tag{2.13}$$

Based on Equation 2.6, this parameter-free approach (*i. e.* which does not involve the learning rate parameter), leads to the following formula for the weight update:

$$\Delta\boldsymbol{\theta}_i^t = \frac{\boldsymbol{g}_i^t}{\frac{\delta^2\mathcal{L}}{\delta\boldsymbol{\theta}_i^{t2}}}. \tag{2.14}$$

The Newton's method provides a better approximation of the cost function than the first-order approximation used so far in the standard gradient descent algorithm. Moreover, there is not any learning rate to tune in the equation; the step is given by the inverse of the Hessian. However, this approach requires a cost function twice differentiable and the involved computations are not tractable for training on real tasks. As a consequence, one must use an optimizer which includes a learning rate, which involves many issues.

Let's take an easy example with a convex function and only one weight w. Figure 2.8 shows different training scenarii, the red arrow indicating the evolution of the weights through the different updates. If the learning rate is too small, the convergence is slow. If the learning rate is too big, it prevents the convergence. The adequate learning rate should lead to convergence in few iterations.



(a) Learning rate too small. (b) Adequate learning rate. (c) Learning rate too big.

Figure 2.8: Learning rate importance.

As one can see, convergence is inherently dependent on finding the appropriate learning rate. This is why one can improve the convergence by choosing an appropriate optimizer.

This optimizer can be combined with a learning rate scheduler. Other techniques, such as curriculum strategies and teacher forcing can be used at training time with the same goal.

2.1.2.1 Optimizers

Optimizers are algorithms that update the model's weights in order to minimize the cost function. The first choice is the weight update frequency *i. e.* the number of samples to process between each weight update. As mentioned previously, samples are grouped into mini-batches characterized by their size. The vanilla approach is Batch Gradient Descent (BGD). It consists in updating the weights after each epoch (once all the samples of the training dataset have been processed). It can lead to slow convergence since there is only one update per epoch. In addition, the whole dataset must be kept in memory, so one may not be able to use this technique in practice depending on the size of the dataset. A variant is Stochastic Gradient Descent (SGD), where the update is performed after each sample $(\boldsymbol{x}, \boldsymbol{y})$. The updates are more frequent but it is computationally expensive. Mini-batch Gradient Descent (MGD) is the balance between the BGD and SGD: the dataset is randomly split into sub-datasets after each epoch, known as mini-batches. The update of the weights occurs after processing each mini-batch. MGD is the most used gradient descent technique.

Another important key point in optimizers is about the learning rate, or more generally, how much to change the current weights with respect to the gradient. Indeed, a too small learning rate will lead to slow convergence and a too high learning rate will prevent reaching the minimum.

The naive approach is to use a fixed learning rate through the epochs, leading to poor performance. Intuitively, one would need a fairly high learning rate at the beginning to quickly approach the global minimum, and then a lower learning rate as it gets closer to it, to avoid diverging.

A first proposition was adding momentum to SGD [17]. The idea is to retain the direction of the previous update, combining it to the new computed gradient. This way, it speeds up convergence and reduces oscillations. Equation 2.6 becomes:

$$\Delta\boldsymbol{\theta}_i^{\boldsymbol{t}} = \eta \boldsymbol{g}_i^{\boldsymbol{t}} + \nu \Delta\boldsymbol{\theta}_i^{\boldsymbol{t-1}}, \tag{2.15}$$

where ν is the momentum, a fixed hyperparameter.

In 2011, the Adagrad (Adaptive gradient) algorithm [18] was proposed with the idea of a custom learning rate per weight:

$$\Delta\boldsymbol{\theta}_i^{\boldsymbol{t}} = \frac{\eta}{\sqrt{\boldsymbol{G}_i^{\boldsymbol{t}} + \epsilon}} \cdot \boldsymbol{g}_i^{\boldsymbol{t}}, \tag{2.16}$$

where $\boldsymbol{G}_i^{\boldsymbol{t}}$ is the sum of the square of the gradients up to t for the weight $\boldsymbol{\theta}_i$. This way, it performs larger updates for parameters whose past gradients were small. The drawback of this method is that the denominator keeps increasing during training, leading to infinitesimally small weight updates.

To solve this problem, G. Hinton proposed in his lecture notes RMSProp (Root Mean Squared Propagation). It consists in replacing $\boldsymbol{G}^{\boldsymbol{t}}$ by $\boldsymbol{D}^{\boldsymbol{t}}$, the exponentially decaying average of previous squared gradients:

$$\boldsymbol{D}^{\boldsymbol{t}} = \nu_D \boldsymbol{D}^{\boldsymbol{t-1}} + (1 - \nu_D)\boldsymbol{g}^{\boldsymbol{t}^2}. \tag{2.17}$$

The authors of [19] noted that there is a unit mismatching issue in the update rule of the previously mentioned algorithms (Adagrad, SGD with momentum, RMSProp). They proposed Adadelta to tackle this problem. They added an exponentially decaying average of squared parameter update term:

$$\boldsymbol{E^t} = \mu_E \boldsymbol{E^{t-1}} + (1 - \mu_E)\Delta\boldsymbol{\theta^t}^2. \tag{2.18}$$

Equation 2.6 becomes:

$$\Delta\boldsymbol{\theta}_i^{\boldsymbol{t}} = \frac{\sqrt{\boldsymbol{E}_i^{\boldsymbol{t-1}} + \epsilon}}{\sqrt{\boldsymbol{D}_i^{\boldsymbol{t}} + \epsilon}} \cdot \boldsymbol{g}_i^{\boldsymbol{t}}. \tag{2.19}$$

One can note that Adadelta does not require any learning rate hyperparameter. In 2015, Adam [20] (Adaptive moment estimation) was proposed as an evolution of the previous algorithms. It is also based on exponentially decaying averages, as for RMSProp and Adadelta. But this time, it is computed for both past squared gradients ($\boldsymbol{v^t}$) and past gradients ($\boldsymbol{m^t}$):

$$\boldsymbol{m^t} = \beta_1 \boldsymbol{m^{t-1}} + (1 - \beta_1)\boldsymbol{g^t}, \tag{2.20}$$

$$\boldsymbol{v^t} = \beta_2 \boldsymbol{v^{t-1}} + (1 - \beta_2)\boldsymbol{g^t}^2. \tag{2.21}$$

The authors noted that $\boldsymbol{m^t}$ and $\boldsymbol{v^t}$ are biased toward zero due to the initial values $\boldsymbol{m^0} = \boldsymbol{v^0} = \boldsymbol{0}$. They computed $\hat{\boldsymbol{m}}^{\boldsymbol{t}}$ and $\hat{\boldsymbol{v}}^{\boldsymbol{t}}$ to counteract these biases:

$$\hat{\boldsymbol{m}}^{\boldsymbol{t}} = \frac{\boldsymbol{m^t}}{1 - \beta_1^t}, \tag{2.22}$$

$$\hat{\boldsymbol{v}}^{\boldsymbol{t}} = \frac{\boldsymbol{v^t}}{1 - \beta_2^t}. \tag{2.23}$$

It leads to the following equation:

$$\Delta\boldsymbol{\theta}_i^{\boldsymbol{t}} = \frac{\eta}{\sqrt{\hat{\boldsymbol{v}}_i^{\boldsymbol{t}} + \epsilon}} \cdot \hat{\boldsymbol{m}}_i^{\boldsymbol{t}}. \tag{2.24}$$

Adam is one of the most used optimizers nowadays. Since then, many alternatives based on this algorithm have been proposed such as AdamW [21], Nadam [22] or AdaMax [23].

On top of these optimizers, one can also add an additional algorithm to control the evolution of the initial learning rate η, which becomes η^t. These algorithms are known as learning rate schedulers.

2.1.2.2 Learning rate schedulers

Learning rate schedulers can be used with any optimizer, even adaptive ones like Adam, as soon as they integrate an initial learning rate. The aim is to have a dynamic learning rate between the successive updates of the weights. For instance, it can enable to have a globally decreasing learning rate, even if there is one learning rate per parameter in the case of adaptive optimizers. Learning rate schedulers can vary the learning rate in many ways. In the following, we describe the most popular learning rate schedulers; they are depicted in Figure 2.9. The simplest is the linear decay:

$$\eta^{t} = \eta^{t-1} - \gamma, \tag{2.25}$$

where γ is a constant step to decrease the learning rate. t can count the number of weight updates or the number of epochs, depending on what one wants to achieve. One can linearly vary the γ term, so that the learning rate decreases slower and slower.

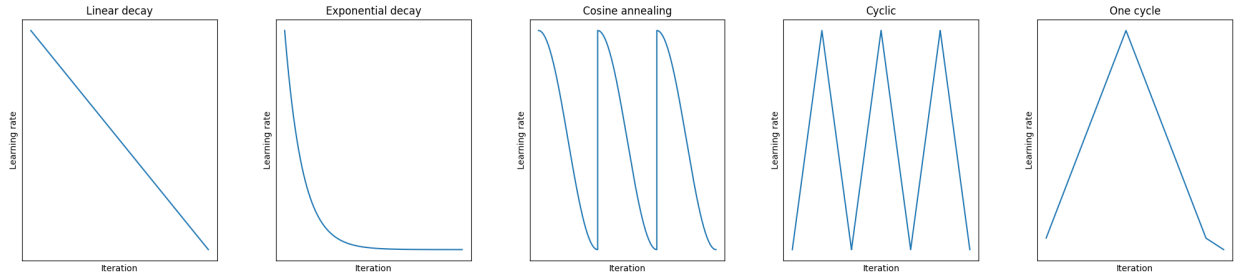


Figure 2.9: Learning rate schedulers.

Another learning rate scheduler consists in exponentially decaying the learning rate:

$$\eta^{t} = \gamma \cdot \eta^{t-1}. \tag{2.26}$$

These two learning rate schedulers are strictly decreasing, mainly to avoid diverging and to reach a minimum. One can define a minimum value for the learning rate or a maximum number of learning rate updates to prevent the learning rate to become too small. In 2017, the authors of [24] proposed the idea of warm restarts in order to avoid getting stuck in local minima or saddle points. They introduce the cosine annealing scheduler, with warm restarts *i. e.* the learning rate decreases following a cosine curve. It gets back to its initial value after T iterations, T being a hyperparameter.

The same year, the authors of [25] proposed a cyclic learning rate scheduler with the same warm restart idea: the learning rate varies linearly between a minimum value and maximum value, in a cyclic way. The next year, a one-cycle policy was proposed [26], improving by far the convergence. It follows the same concept of linear variation between lower and upper bound, but this is done only once. Then, when the cycle is ended, the learning rate is annealed during the last iterations, with an important decrease. These two last approaches are based on a range test approach to determine the bounded values to choose: a first training is carried out by increasing the learning rate after each weight update, leading to diverging and thus an increasing loss. The maximum learning rate is generally the learning rate before the loss increase. The minimum learning rate is taken as one tenth of the maximum.

Learning rate schedulers can also be chained together, to have a dynamic strategy during training, or be adapted to the evolution of the metrics for instance. Learning rate schedulers can enable to reach better results or simply to converge more efficiently. However, one can note that in most papers, the authors use the Adam optimizer without any learning rate scheduler. Indeed, these schedulers often need to tune some hyperparameters, sometimes very sensitive to the model or the data used. To summarize, adaptive optimizers, like Adam, allow to guarantee a certain convergence without needing to set hyperparameters in a precise

way, it is very useful in the research phase. However, for an industrial application, it may be relevant to take the time to find the right hyperparameters for the use of a learning rate scheduler to reach even better performances.

2.1.2.3 Curriculum strategies

Curriculum strategies are inspired by the way humans learn. Curriculum learning is the most known strategy. It consists in learning a task with an increasing difficulty. For instance, one first learns to write isolated letters, words, then sentences before writing whole dissertations.

In [27], the authors show improving results when gradually increasing the difficulty on two different tasks: language modeling, with an increasing vocabulary, and shape recognition, with an increasing shape variability. This idea of curriculum learning is very related to the task and must be adapted on a case-by-case basis.

Curriculum paradigm can also be applied to Dropout for instance: in [28], the authors proposed to slightly increase the probability of dropout τ during training:

$$\tau^{\boldsymbol{t}} = (1 - \bar{\tau}) \exp(-\gamma t) + \bar{\tau}, \gamma > 0, \tag{2.27}$$

where $\bar{\tau}$ is the final dropout rate, t is the number of iterations and $\gamma = \frac{1}{T}$ (T being the total estimated number of weight updates during training). This dropout scheduler showed promising results on various image classification datasets.

2.1.2.4 Teacher forcing

Teacher forcing is another strategy, applied at training time, to improve the convergence speed of neural networks. This is used for tasks in which the expected output is a sequence of tokens $\boldsymbol{y}$ which are recurrently predicted *i. e.* that the prediction of $\hat{\boldsymbol{y}}^{\boldsymbol{t}}$ depends on the previous predicted tokens $\hat{\boldsymbol{y}}^{\boldsymbol{t-i}}$, $i \in [0, t-1]$. The idea is to provide the ground truth tokens instead of the previous predicted tokens to predict the next token. It has two main advantages. First, the computations can be parallelized during training to predict the whole expected sequence at once, thus reducing the training time per epoch. Second, if one considers training from scratch, the predictions are very poor at the beginning. Giving the ground truth tokens instead of the poor predictions enables the model to make predictions based on realistic previous tokens, reducing its prediction errors and improving the convergence.

However, teacher forcing should not be used all the time. Indeed, if one always provides the ground truth as previous predicted tokens, the model only sees perfect context to make the next prediction and is not robust: it does not generalize at prediction time, when some errors may occur. One way to alleviate this issue is to introduce some errors in the provided ground truth in order to train the model to predict correct tokens even when its previous predictions are wrong. One can also use teacher forcing for the first epochs only, and then train the model with its own predictions in a second step.

Now that we have seen how to handle the generalization and convergence problems, we can now focus on the vanishing and exploding gradient issues.

2.1.3 Dealing with vanishing and exploding gradients

Vanishing and exploding gradients [29] are phenomenons appearing in the lowest layers of deep networks. It is induced by the chained rule used during backward propagation (Equation 2.8 and 2.9). Indeed, based on those equations, the formula for the gradient computation for the weights of the lowest layer $\boldsymbol{W}_1$ is:

$$\frac{\delta \mathcal{L}}{\delta \boldsymbol{W}_1^t} = \frac{\delta \mathcal{L}}{\delta \boldsymbol{H}_1} \cdot \frac{\delta \boldsymbol{H}_1}{\delta \boldsymbol{W}_1^t} = \frac{\delta \mathcal{L}}{\delta \boldsymbol{H}_\gamma} \cdot \left(\prod_{i=2}^{\gamma} \frac{\delta \boldsymbol{H}_i}{\delta \boldsymbol{H}_{i-1}} \right) \cdot \frac{\delta \boldsymbol{H}_1}{\delta \boldsymbol{W}_1^t}. \tag{2.28}$$

It includes a multiplication in chain implying more and more terms as one approaches the lowest layers of the network. It is at the origin of the well-known vanishing and exploding gradient issues:

- if $0 \leq |\frac{\delta \boldsymbol{H}_i}{\delta \boldsymbol{H}_{i-1}}| < 1, \forall i$, the gradient will tend to zero: this is the vanishing gradient. It means that the weights are not updated anymore for the lowest layers.
- if $|\frac{\delta \boldsymbol{H}_i}{\delta \boldsymbol{H}_{i-1}}| > 1, \forall i$, the gradient will tend to infinity: this is the exploding gradient. It will lead to numerical instabilities and overflows, due to hardware limitations.

$\frac{\delta \boldsymbol{H}_i}{\delta \boldsymbol{H}_{i-1}}$ corresponds to the gradient of the output of layer i with respect to its input *i. e.* the gradient of the elementary functions. The main used functions are activation functions and linear layers.

For an activation function f, there is no trainable weights involved:

$$\frac{\delta \boldsymbol{H}_i}{\delta \boldsymbol{H}_{i-1}} = f'(\boldsymbol{H}_{i-1}). \tag{2.29}$$

Linear layers are layers in which each neuron is computed as a weighted sum of the previous neurons: $\boldsymbol{H}_i = \boldsymbol{W}_i \boldsymbol{H}_{i-1}$. Densely-connected layers, fully-connected layers and formal neuron are different names used for linear layers. Its associated gradient is:

$$\frac{\delta \boldsymbol{H}_i}{\delta \boldsymbol{H}_{i-1}} = \boldsymbol{W}_i. \tag{2.30}$$

To sum up, one must pay attention to the values of the activation functions, and more precisely to their derivatives, as well as those of the trainable weights. In the following, we demonstrate how the choice of the activation functions and of the weight initialization method is important. We also present some other techniques to avoid vanishing and exploding gradient issues: normalization and residual connections.

2.1.3.1 Activation functions

In neural networks, each neuron layer is followed by an activation function (no activation can be seen as identity function). Since real-world tasks are complex, linear activation functions

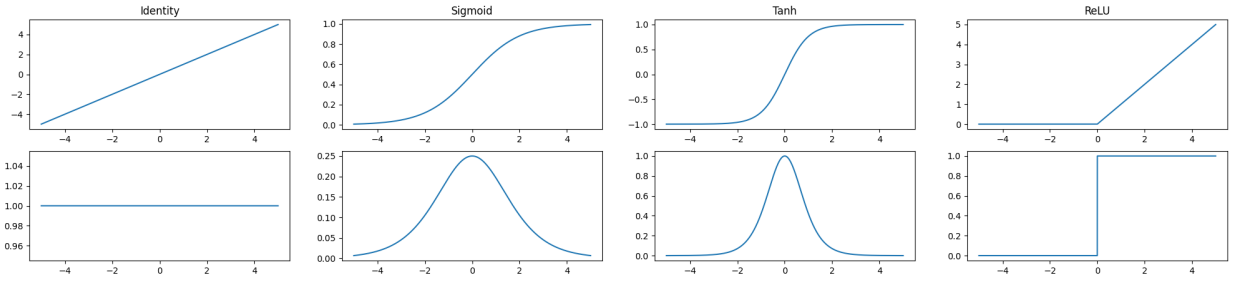


Figure 2.10: Activation functions (top) and their associated derivatives (bottom).

are not sufficient to model adequate functions. This way, non-linear activation functions are mainly used such as tanh, sigmoid or ReLU.

Figure 2.10 depicts the main activation functions and their corresponding derivatives. As one can note, tanh and sigmoid derivatives lead to bounded values between 0 and 1, and 0 and 0.25, respectively. As we have seen previously, this will lead to vanishing gradient issues. The impact grows with the number of layers in the network. ReLU activation was introduced to tackle this problem, its derivative being either 0 or 1.

Another important activation function is softmax, which enables to get a probability distribution over an input vector $\boldsymbol{x} = [\boldsymbol{x}_1, ..., \boldsymbol{x}_n]$:

$$\text{softmax}(\boldsymbol{x}_i) = \frac{\exp(\boldsymbol{x}_i)}{\sum_{j=1}^{n} \exp(\boldsymbol{x}_j)}. \tag{2.31}$$

It is notably used as last activation function, as a decision layer, in the case of classification problems.

2.1.3.2 Weights Initialization

The initialization of the weights is a crucial element regarding convergence. Let's consider an FFN. A naive approach would be to initialize all weights to zero, or with a constant value. If one follows this approach, it would lead to each neuron of a same layer computing the exact same value, since their associated weights and their inputs are identical. The associated gradients would also be identical leading to all neurons of a same layer learning the same features. Training would not be efficient at all.

To tackle this problem, one can use a normal $\mathcal{N}(0, \sigma^2)$ or uniform $\mathcal{U}(-u, u)$ distribution to bring randomness, and make the different neurons behave differently, thus learning different features. But one should keep in mind that too small weights will lead to vanishing gradient issues, and too high weights to exploding gradient issues, as explained before. A solution is then to choose a weight initialization that enables to preserve the variance from one layer to another, but this depends on the activation functions used.

In 2010, Xavier Glorot *et. al.* proposed in [30] some hyperparameters for uniform and normal distributions to preserve the variance from one layer to another when using tanh or

sigmoid activation functions:

$$u = \sqrt{\frac{6}{n_{\text{in}} + n_{\text{out}}}}, \tag{2.32}$$

and

$$\sigma = \sqrt{\frac{2}{n_{\text{in}} + n_{\text{out}}}}, \tag{2.33}$$

where n_{in} and n_{out} are the numbers of neurons in the previous layer and in the next layer, respectively.

In 2015, Kaiming He *et al.* proposed in [31] another hyperparameter for the use of the ReLU activation, in the same spirit of preserving the variance from one layer to another:

$$\sigma = \sqrt{\frac{2}{n_{\text{in}}}}. \tag{2.34}$$

2.1.3.3 Normalization

Normalization aims at fixing the distribution of the output of a layer by normalizing the values *i. e.* to have 0 mean and unit variance. It is a good way to reduce vanishing gradient when using sigmoid or tanh activation functions for example. As a matter of fact, based on Figure 2.10, 0 mean and unit variance force the derivative of sigmoid and tanh to be far from their 0-boundary: the associated gradient values are bigger, reducing the vanishing gradient issue induced by the chain rule (Equation 2.28). The authors of [32] proposed Batch Normalization to also tackle the internal covariate shift issue. It is defined in [32] as the modification in the distribution of network activations due to the update of the network weights during training.

Given a set of values $\mathcal{X} = \{x_1, ..., x_N\}$, normalization consists in computing $\hat{\mathcal{X}} = \{\hat{x}_1, ..., \hat{x}_N\}$ with $\hat{\mathcal{X}} \sim \mathcal{N}(\mu_{\hat{X}}, \sigma^2_{\hat{X}})$ so that $\mu_{\hat{X}} = 0$ and $\sigma^2_{\hat{X}} = 1$. The normalization is carried out in three steps:

$$\mu_X = \frac{1}{N}\sum_{i=1}^{N} x_i, \tag{2.35}$$

$$\sigma^2_X = \frac{1}{N}\sum_{i=1}^{N}(x_i - \mu_X)^2, \tag{2.36}$$

$$\hat{x}_i = \frac{x_i - \mu_X}{\sqrt{\sigma^2_X + \epsilon}}, \tag{2.37}$$

where ϵ is an arbitrary positive hyperparameter close to zero, to avoid a division by zero. An additional step is proposed in this paper to scale and shift the obtained normalized values respectively by γ and β, two trainable weights:

$$y_i = \gamma\hat{x}_i + \beta \tag{2.38}$$

Given an input of shape (N, L, C) where N is the mini-batch size, L is the sequence length ($L = H \times W$ for an image) and C is the number of channels, Batch Normalization consists

in applying normalization through the samples of a same mini-batch, but considering the channels independently. Batch Normalization is widely used nowadays; it showed interesting properties such as quicker convergence and regularization, especially for large mini-batches. It also avoids values to grow too much through the layers in the case of ReLU activation for instance. However, it comes at the cost of additional computations, impacting prediction time for example. At training time, the loss due to the additional computations may be compensated by the better convergence.

Running mean and variance is usually computed during training and used at inference time to preserve an adequation between training and prediction. Otherwise, distribution differences between these two configurations could lead to poorer results.

In 2018, the authors of [33] showed that the efficiency of the batch normalization is not mainly due to its impact on the internal covariate shift. They suggest that it would be due to its reparametrization of the optimization problem making it more stable and smoother in the sense of loss Lipschitzness and loss β-smoothness respectively.

In 2016, other normalization techniques have been proposed: Instance Normalization [34] and Layer Normalization [35]. They follow the same idea; the key difference is the dimensions on which normalization is performed. Instance Normalization consists in normalizing over the L axis, considering the different channels and the different samples independently. Layer Normalization applies normalization over the channels and the L axis, still considering the samples independently. More recently, Group Normalization [36] has been proposed as an in-between, grouping some channels together for the normalization. Normalization techniques are visually compared in Figure 2.11.

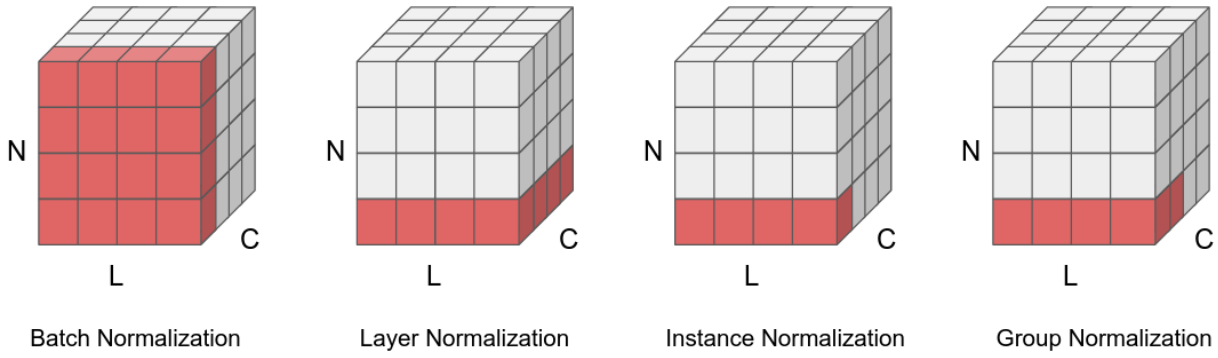


Figure 2.11: Visualization of the different normalization techniques

2.1.3.4 Residual connections

Residual connections (or skip connections) have been introduced in [37], in which the authors proposed the ResNet. A residual connection is a way to introduce features from other part of the network than just the previous layer. Figure 2.12 provides an illustration of the concept of residual connection. As one can note, $\boldsymbol{H}_{i-1}$ is followed by two sub-pipelines before being merged through an operation. This operation is usually an element-wise addition but it can also be a concatenation or an element-wise multiplication for instance.

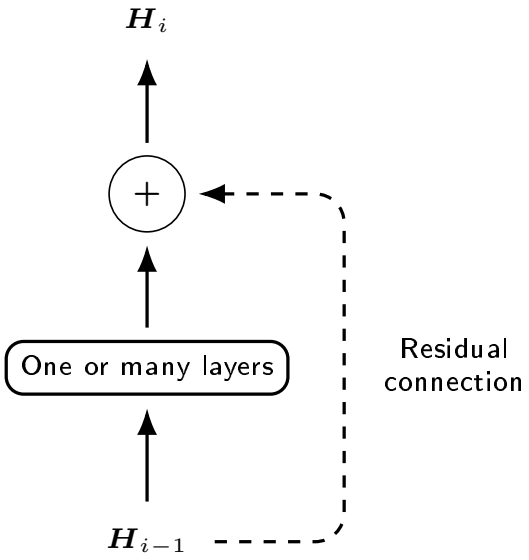


Figure 2.12: Residual connection between hidden state $\boldsymbol{H}_{i-1}$ and $\boldsymbol{H}_i$.

The use of residual connections bring two main advantages. First, it aggregates information from different layers, leading to a multi-scale feature extraction, which can lead to better results. Second, it helps reducing the vanishing gradient. Let us take the example of the element-wise addition as merging operation between layer $\boldsymbol{H}_{i-1}$ and $\boldsymbol{H}_i$, and identity function as second branch. Equation 2.9 becomes:

$$\frac{\delta \mathcal{L}}{\delta \boldsymbol{H}_{i-1}} = \frac{\delta \mathcal{L}}{\delta \boldsymbol{H}_i} \cdot \left(\frac{\delta \boldsymbol{H}_i}{\delta \boldsymbol{H}_{i-1}} + 1 \right). \tag{2.39}$$

As one can note, it avoids the gradient from decreasing from one layer to another during backpropagation.

2.1.3.5 Hardware limitations

Deep Neural Networks are trained with computers, leading to some limitations. Computation time for forward and backward passes are dependent on the hardware used. Modern frameworks (Tensorflow, Pytorch) provide highly optimized algorithms for commonly used operations, notably on Graphic Processing Unit (GPU). Nonetheless, as networks get deeper and deeper and datasets get larger and larger, it can be complicated to reach convergence in a reasonable training time or to use large mini-batches. One way to reduce this issue is to use the concept of automatic mixed precision *i. e.* to perform the operations with 16-bit floats instead of 32-bit floats, when the operation is sufficiently numerically stable. Indeed, it enables to reduce both computation time and GPU memory consumption in many cases.

As mentioned previously, exploding gradient induces overflows, due to the limitation of 32-bit floats encoding. One way to counteract this issue is to use gradient clipping. It consists in setting a maximum value that the gradient values cannot exceed.

We showed that neural networks can estimate complex functions, through gradient descent algorithms. We presented the main components and techniques to deal with the generalization and convergence issues as well as the vanishing and exploding gradients. All these techniques enable to have very powerful, stable and deep neural networks, explaining their impressive performance in many tasks. We now study the main components and architectures which enables to tackle computer vision and sequence-to-sequence problems, which will enable us to understand how to handle image-to-sequence tasks such as HTR.

2.2 HTR: a computer vision problem

Computer Vision (CV) refers to the various techniques that enable computers to see and understand the content of images or videos. It includes numerous applications such as image captioning, object detection, facial recognition or image classification. Therefore, HTR naturally falls into the computer vision field. Indeed, in HTR, we aim at recognizing the text from an input image $\boldsymbol{X}$ of shape (H, W, C). The model must learn to extract relevant features from the input images in order to detect the different characters. We limit ourselves here to study the evolution of the deep learning practices and architectures which made it possible to improve the performances for these computer vision tasks.

2.2.1 Multi-Layer Perceptron for computer vision

MLP is one of the first neural network proposed to tackle computer vision problems. However, MLP presents a main drawback. The input needs to be of fixed size since each neuron of a given layer is connected to all neurons of the previous layer (the input layer included). This way, the number of trainable weights is dependent on the input size. It implies another drawback: the number of weights exponentially increases with the input size. Considering our input image $\boldsymbol{X}$, it would require $H \times W \times C \times n$ weights for a first neuron layer containing n neurons. This is shown in Table 2.1 for different input sizes, resolutions (number of pixels per inch) and color encodings, and for a first layer containing 1028 neurons (taken arbitrarily). As one can note, the number of weights is really high (26.8G) for a A4 RGB input with a resolution of 300 DPI, which represents the majority of paper documents used with standard scanning quality. Decreasing the input size to A6 format (4 times smaller than A4), and the resolution to 100 DPI, leads to 38 times less parameters required. However, it still represents a high number of trainable parameters, just to consider the first layer.

MLP have been successfully applied for classification tasks on toy datasets such as MNIST [38] which consists in black-and-white handwritten digit images of 28 pixels by 28 pixels. Nonetheless, this relation between weights and input size prevents the usage of MLP for the majority of computer vision tasks, when the input images are large and/or cannot be compressed due to the importance of low-level information.

2.2.2 Convolutional Neural Network

In 1989, LeCun *et al.* [39] proposed the first Convolutional Neural Network (CNN), LeNet-5, based on the Neocognitron [40] of Fukushima *et al.*. It was inspired by discoveries related to

Format	Resolution	Color	H	W	C	n	# weights
A4 (210x297 mm)	300 DPI	RGB	3,508	2,480	3	1028	26,8G
		Gray-scaled	3,508	2,480	1	1028	8.9G
	100 DPI	RGB	1,170	827	3	1028	3.0G
		Gray-scaled	1,170	827	1	1028	1.0G
A6 (105x148 mm)	300 DPI	RGB	1,754	1,241	3	1028	6.7G
	100 DPI	RGB	585	414	3	1028	0.7G
MNIST	✗	B&W	28	28	1	1028	0.8M

Table 2.1: Number of weights implied by the first layer of an MLP with respect to the input.

visual mechanisms in living organisms. CNN corresponds to MLP in which some convolutional and pooling layers are added, usually at the beginning of the network. Convolutions are now the key element of almost all neural networks applied to CV tasks. A CNN applied to image classification is depicted in Figure 2.13.

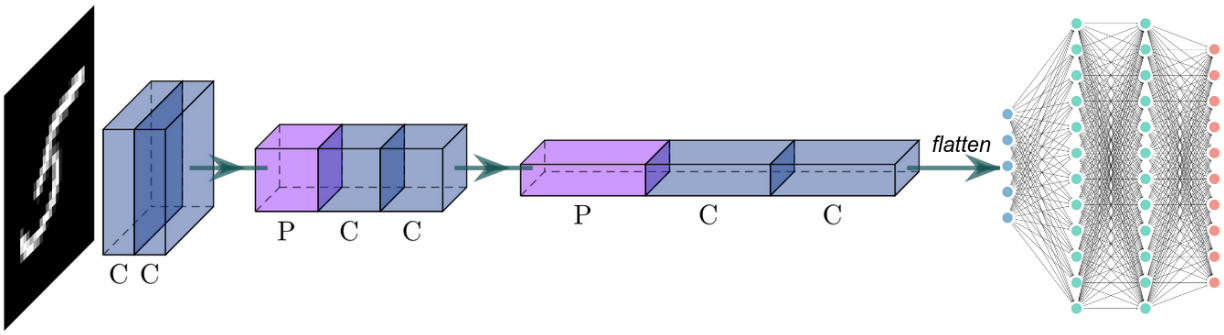


Figure 2.13: Convolutional Neural Network. C: convolutional layer. P: pooling layer.

2.2.2.1 Convolutional layers

A convolutional layer consists in applying n_k weighted sums between the elements of n_k kernels and the input, through a sliding window over the input. A convolution is defined by the size of the kernels $k_\text{h} \times k_\text{w}$, their stride $s_\text{h} \times s_\text{w}$, which corresponds to the displacement step of a sliding window on the two dimensions and their dilation factor $d_\text{h} \times d_\text{w}$, which corresponds to the spacing between two consecutive values to be considered in the calculation. These different parameters are represented through an example in Figure 2.14. The convolution operation is represented by the operator $\odot$.

Mathematically, for a single kernel $\boldsymbol{k}$:

$$\boldsymbol{Y} = \boldsymbol{X} \odot \boldsymbol{k}, \tag{2.40}$$

with

$$\boldsymbol{Y}_{i,j} = \sum_{h=1}^{k_\text{h}} \sum_{w=1}^{k_\text{w}} \sum_{c=1}^{C} \boldsymbol{X}_{s_\text{h}*(j-1)+d_\text{h}*(h-1)+1, s_\text{w}*(i-1)+d_\text{w}*(w-1)+1, c} \boldsymbol{k}_{h,w,c} \tag{2.41}$$

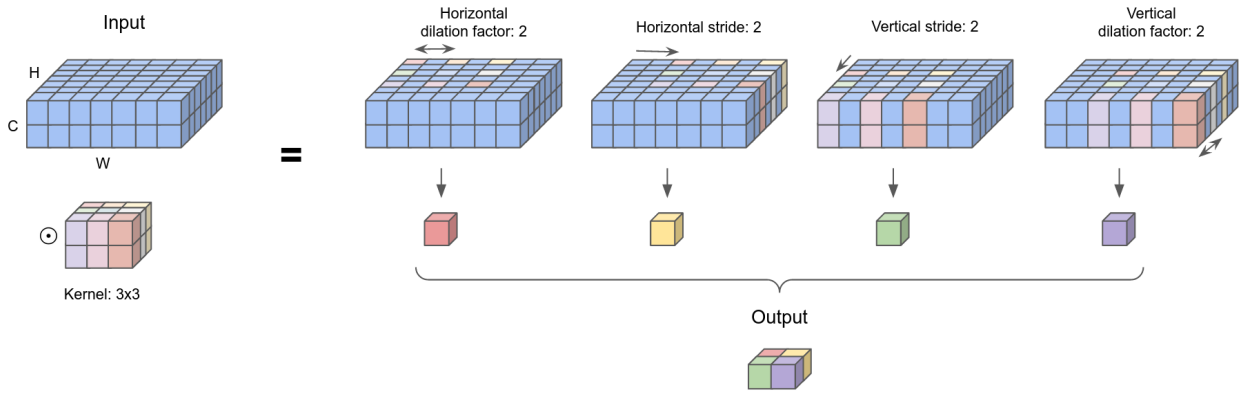


Figure 2.14: Convolution applied on a 2-channel input with a single 3×3 kernel, 2×2 stride and 2×2 dilation factor.

The standard configuration is as follows: 3×3 kernel, 1×1 dilation factor, 1×1 stride. An extension to 2 kernels with this standard configuration is depicted in Figure 2.15. Increasing the stride mainly aims at reducing the size of the tensors and thus the memory consumption. The dilation factor can be increased to capture relationships over a longer distance or to model higher level elements using fewer convolutional layers. Padding can also be added to preserve the original size of the input image.

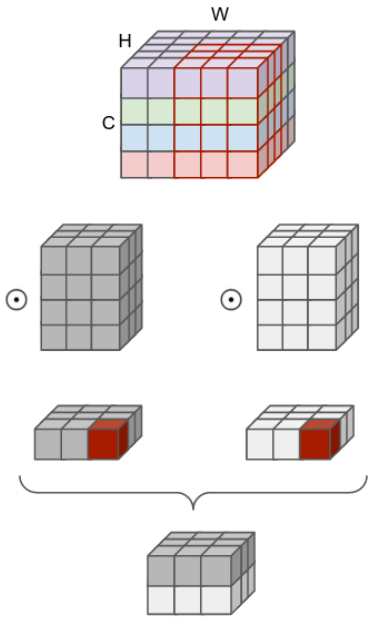


Figure 2.15: Convolution applied to a 4-channel input with 2 kernels of size 3×3.

As one can note, convolutions imply local connectivity: the output at a specific position only depends on neighboring inputs. In addition, the same kernel is applied at different locations of the input: this is a property of convolutional layers called weight sharing. It assures shift-invariance *i. e.* a same pattern will be recognized in the same way no matter where it is in the image. It also enables to dramatically reduce the number of trainable

parameters when compared to a dense layer. Indeed, contrary to fully-connected layers, the number of weights induced by a convolutional layer does not depend on the width and the height of the input. Considering the input image $\boldsymbol{X}$, a convolution applying n_{k} kernels of size $k_{\text{H}} \times k_{\text{W}}$ implies $C \times k_{\text{H}} \times k_{\text{W}} \times n_{\text{k}}$ weights. Well-known CNN architectures such as ResNet [37] are made up of convolutions using up to 512 channels, sometimes more. A convolution applied on a 1028-channel input with 1028 kernels of size 3×3 would then lead to 9.5M of weights, no matter the input image dimensions. This is far from the billions of parameters required for one fully-connected layer applied on an A4 input image (Table 2.1). But contrary to the convolutional layer, each neuron of a fully-connected layer take advantage of the global context of the whole input.

Stacking several convolutional layers while maintaining a large number of channels still leads to a huge number of trainable weights. One way to alleviate this issue is to use Depthwise Separable Convolutions (DSCs) [41] instead of the standard convolutions. This kind of convolutions is depicted in Figure 2.16.

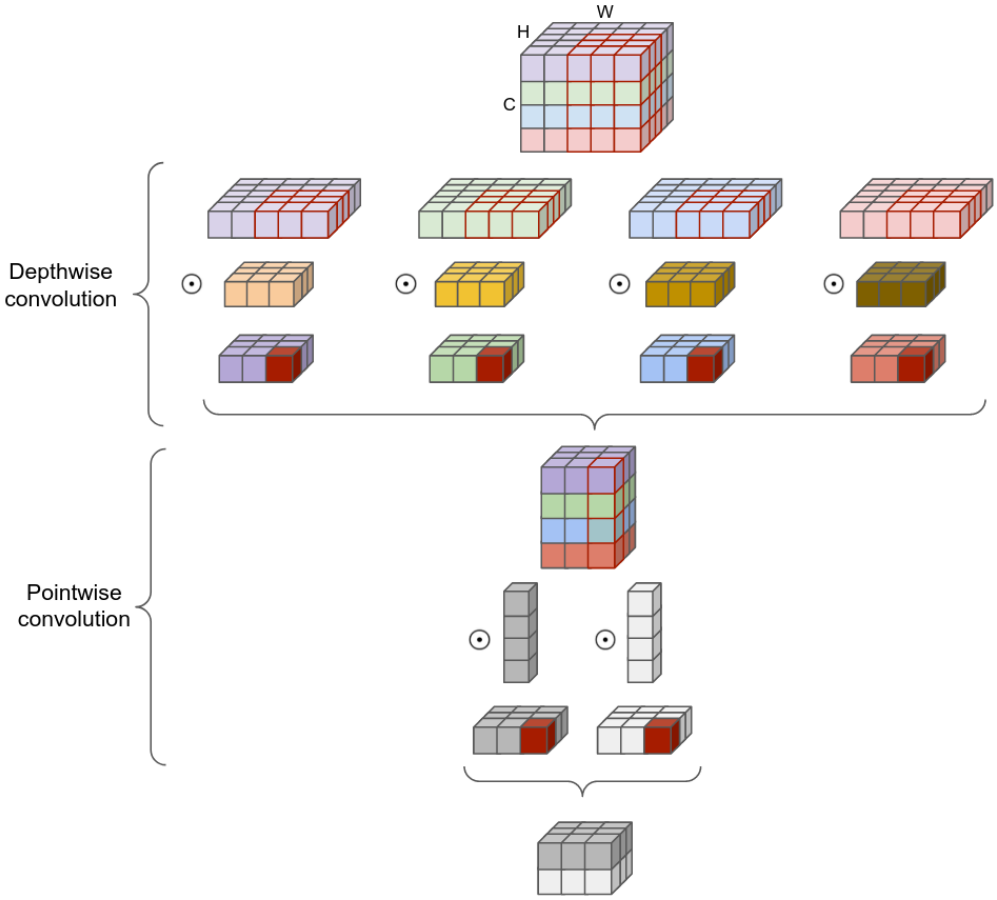


Figure 2.16: Depthwise Separable Convolution applied to a 4-channel input with 2 kernels of size 3×3.

DSC is a composition of two convolutions:

- Depthwise convolution: each of the C channels of the input is treated independently by a specific kernel of shape $k_{\text{H}} \times k_{\text{W}} \times 1$, leading to C features maps which are merged

together. This first step is not sufficient since it would be to consider that the channels are independent, which is false when we speak of color channels red, green and blue of an image for example.

- Pointwise convolution: n_{k} kernels of shape $1 \times 1 \times C$ are applied on these intermediate feature maps, leading to the final results. This second operation aims at modeling the relations between the different channels.

Then, the associated number of weights is $k_{\mathrm{H}} \times k_{\mathrm{W}} \times C$ for the depthwise convolution and $C \times n_{\mathrm{k}}$ for the pointwise convolution. It results in $C \times (k_{\mathrm{H}} \times k_{\mathrm{W}} + n_{\mathrm{k}})$ parameters for the global operation. We can now compare the difference with standard convolution for different parameters through Table 2.2.

C	n_{k}	k_{H}	k_{W}	# weights convolution $C \times k_{\mathrm{H}} \times k_{\mathrm{W}} \times n_{\mathrm{k}}$	# weights DSC $C \times (k_{\mathrm{H}} \times k_{\mathrm{W}} + n_{\mathrm{k}})$
512	512	3	3	2.4M	0.3M
1028	1028	3	3	9.5M	1.1M
1028	1028	5	5	26.4M	1.1M

Table 2.2: Comparison of the number of weights implied by a standard convolution and by a Depthwise Separable Convolution (DSC).

As one can note, using DSC breaks the linearity with respect to the kernel size in terms of number of trainable parameters, leading to a decrease factor of around 9 for the standard kernel size 3×3. This factor is all the more important as the size of the kernels is large.

2.2.2.2 Pooling layers

Pooling layers are the other component of CNN. They consist in downsampling the input so as to decrease the memory consumption, leading to shorter calculation times. To this end, a kernel is applied with stride, similarly to convolutional layers, but there is no trainable parameters involved: the returned value depends on a predefined function such as max or mean. The idea is to compress the input signal by only keeping the relevant information, discarding the non-relevant one. This way, during back-propagation, only gradients of selected values are propagated.

CNN are traditionally made up of a stack of blocks of convolutional and pooling layers, ended by one or multiple fully-connected layers as shown in Figure 2.13. CNN still suffer from the fixed-input-shape requirement induced by those fully-connected layers. Adaptive pooling is a way to alleviate this issue. It consists in a pooling layer with a dynamic kernel in order to have a fixed output shape. It is compared with standard pooling in Figure 2.17, for the max function.

Given the input and output shapes, the kernel size is automatically adapted to cover all the input signal, giving the correct number of values expected as output. This way, adaptive pooling enables to take inputs of variable size, still using fully-connected layers, by fixing intermediate tensor sizes in the network. It enables to preserve the original resolution and ratio of the image for the first layers for tasks implying outputs of fixed size.

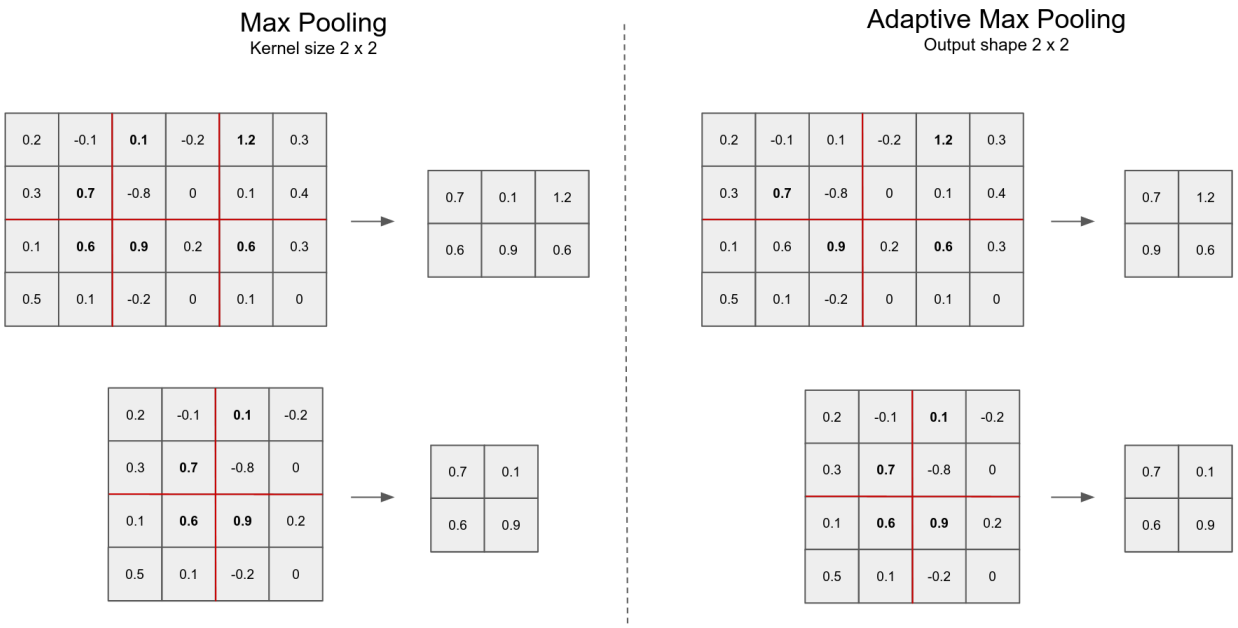


Figure 2.17: Max Pooling compared to Adaptive Max Pooling.

2.2.2.3 Receptive field

As shown previously, convolution and pooling operations compute outputs based only on neighboring inputs *i. e.* each output has a partial and unique view of the input image. The receptive field corresponds, for a given layer, to the size of the region in the input from which each value of this layer has been calculated. In other words, it quantifies what is seen by a given neuron. It is an important parameter to understand the highest level of representation that a neural network is able to model. This notion is represented in Figure 2.18. In this example, the receptive field for layer 2 is (5, 5) for height and width, respectively.

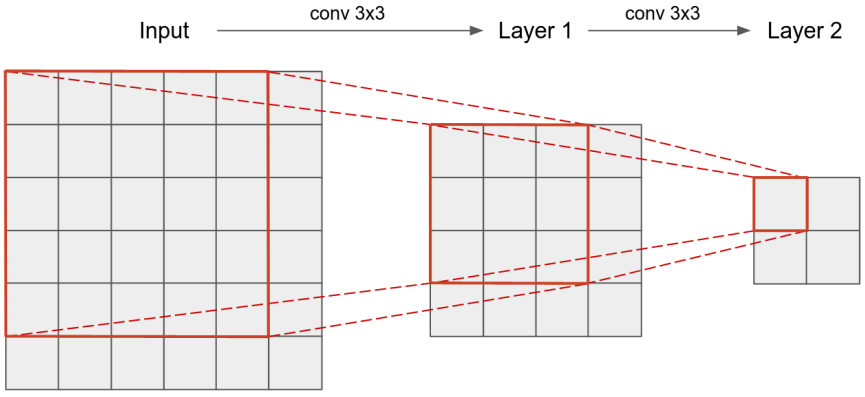


Figure 2.18: Receptive field visualization for 2 convolutions of kernel 3×3, stride 1×1 and dilation 1×1.

The receptive field is fixed for strictly sequential feed-forward neural networks. However, when one uses residual connections, multiple receptive fields are combined for a same layer,

leading to multi-scale representations.

CNN have gradually become the default architecture for computer vision tasks bringing state-of-the-art results on several tasks. For example, well-known architectures such as AlexNet [42] and VGGNet [43] have been proposed for image classification. However, we have highlighted the limitations of such architectures: even using adaptive pooling techniques, they can only handle tasks for which the size of the output is fixed. There are some tasks for which preserving the original size, or at least the dimension ratio, is important. For instance, most segmentation tasks imply pixel-to-pixel classification, which requires to preserve the image size. As a consequence, CNN cannot be used to solve these tasks.

2.2.3 Fully Convolutional Network

Fully Convolutional Networks (FCNs) have been proposed in response to the need for outputs of variable size. It corresponds to a CNN without any fully-connected layer: it is only made up of convolutional components: convolutions and poolings. This way, the output of the network is the output of the last convolutional layer: the input shape can be preserved from the beginning to the end of the network.

U-Net is a well-known FCN architecture; it is depicted in Figure 2.19. In this architecture, hidden representations are compressed through convolution or pooling operations (blue part) with stride higher than 1×1 to save memory consumption. Then, to get back to the original size of the input image, it relies on different upsampling techniques (beige part).

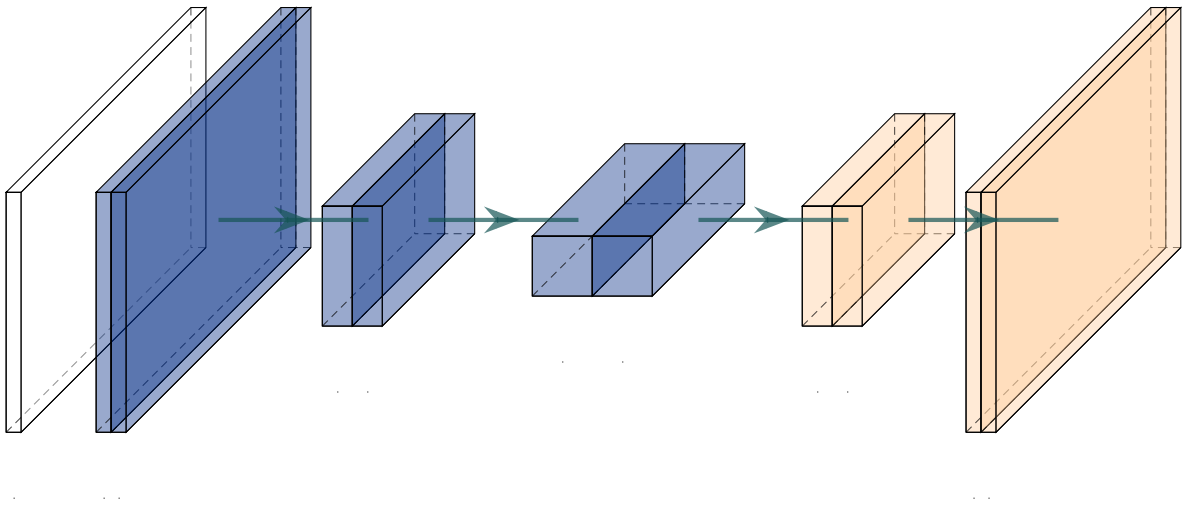


Figure 2.19: Overview of the U-Net architecture.

The naive way to upsample the hidden representation is to locally duplicate the information. One can also use an interpolation based on the values of the neighboring cells (a bi-linear interpolation for example). Another common technique is the inverse operation of max pooling. To this end, the indices of the max values selected during max pooling are kept in memory for the max unpooling operation. They are used to place the values of the current hidden representation; other locations are set to zero.

Transposed convolution consists in applying n_k kernels of size $k_\text{H} \times k_\text{W} \times C$ to each location of an input $\boldsymbol{X} \in \mathbb{R}^{H \times W \times C}$. This way, a $k_\text{H} \times k_\text{W} \times n_k$ representation is computed for each location of the input, before being merged together. In the standard configuration, it leads to an output $\boldsymbol{Y} \in \mathbb{R}^{(H+k_\text{H}-1) \times (W+k_\text{W}-1) \times n_k}$. However, it can also be used with a stride and a dilation factor different from 1×1 to perfectly fit the dimensions of the previous neural layers, no matter the configuration used for the convolutions.

The ability of FCN to process inputs of variable sizes as well as their low number of parameters make them one the most popular model for many computer vision tasks, including segmentation tasks such as semantic segmentation of biomedical images [44] and instance segmentation [45].

As we have seen so far, the computer vision tasks are mainly handled by CNN, notably to reduce the number of parameters of the models when compared to MLP. However, CNN also faces fixed-size input/output issues and their modeling ability is dependent on their receptive field. FCN architectures enable to deal with inputs of variable sizes, but it remains two main issues. To model dependencies across the whole input signal with an FCN, it would require to have a receptive field as big as the input size, which could require many and many layers (and thus many trainable parameters). Moreover, FCN cannot deal with outputs whose size is independent to that of the inputs as is. It makes the computer vision models presented in this section unsuitable for the HTR task which implies output sequences of variable lengths. To solve this issue, we will now study the HTR under the sequence-to-sequence point of view.

2.3 HTR: a sequence-to-sequence problem

There is no consensus about the definition of a Sequence-to-sequence (seq2seq) problem. In this thesis, we refer to sequence-to-sequence problems when inputs and outputs are sequences of variable and independent sizes (no assumption is made about the length of the output sequence with respect to the length of the input sequence). Handwritten Text Recognition must handle images as inputs and sequences of tokens as outputs. It can be considered as a sequence-to-sequence problem. Indeed, contrary to the classification task, where there is globally only one entity present in the image, a handwritten document image contains several characters, ordered horizontally and vertically. The order can be even more complex depending on the layout of the document. Moreover, the image can be seen as a 2D sequence of pixels or a sequence of columns or rows of pixels $\boldsymbol{x}$ of length L_x. This way, the input can be considered as a sequence, and the output is a sequence $\boldsymbol{y}$ of L_y characters: this is a sequence-to-sequence problem.

Sequence-to-sequence tasks imply three main issues:

- Extracting features from the whole input sequence. It requires to understand the relations between the different items of the sequence *i. e.* to model the input dependencies.
- Handling input and output of variable sizes. The architecture must be able to predict sequences of any length, independently of the input length.

- Extracting features from the previous predictions. Modeling the output dependencies enables to take into account the output context for the prediction of the next item.

In the following, we will go over the main components that enable to deal with these issues.

2.3.1 Modeling dependencies

Modeling dependencies aims at extracting features from a whole sequence, by modeling the relations between the different items of this sequence, rather than considering them independently of each other. CNNs enable to model local dependencies from the input, but it does not model the output dependencies. Until the end of the 2010's, state-of-the-art approaches for output sequence modeling were mainly based on Hidden Markov Model (HMM).

2.3.1.1 Hidden Markov Model

HMM are statistical models which aims at capturing hidden information from sequential observations. HMM can be represented as a finite state machine as in Figure 2.20. It is defined by:

- A set of states $S = \{s_1, ..., s_N\}$, and a sequence of L states $\boldsymbol{q} = [\boldsymbol{q^1}, ..., \boldsymbol{q^L}]$ with $\boldsymbol{q^t} \in S$.
- A sequence of observations $\boldsymbol{x} = [\boldsymbol{x^1}, ..., \boldsymbol{x^L}]$, which can be real valued observation vectors.
- An emission model B, which defines the probabilities of an observation $\boldsymbol{x^t}$ given a state i: $p(\boldsymbol{x^t}|\boldsymbol{q^t} = s_i)$. B can either be composed of discrete probabilities or continuous gaussian mixtures.
- A transition model A, which defines the probabilities $\boldsymbol{a}_{i,j}$ of moving from state i to state j: $p(\boldsymbol{q^t}|\boldsymbol{q^{t-1}})$
- An initial probability distribution model Π, which defines the probabilities $\boldsymbol{\pi}_i$ of starting in state i: $p(\boldsymbol{q^1})$.

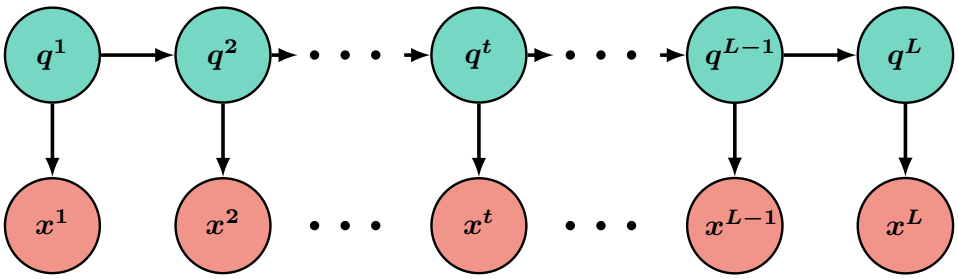


Figure 2.20: Hidden Markov Model.

HMM follow the Markov assumption implying that the state $\boldsymbol{q^t}$ at timestep t only depends on the previous state $\boldsymbol{q^{t-1}}$:

$$p(\boldsymbol{q^t}|\boldsymbol{q^1}, ..., \boldsymbol{q^{t-1}}) = p(\boldsymbol{q^t}|\boldsymbol{q^{t-1}}). \tag{2.42}$$

In a first step, B, A and Π are learned through the forward-backward algorithm (also known as Baum-Welch algorithm) [46], which is a special case of the Expectation-Maximization algorithm [47]. Then, the most likely state sequence $\boldsymbol{q}^*$ given an observed sequence $\boldsymbol{x}$ is obtained using the Viterbi algorithm [48]:

$$\boldsymbol{q}^* = \arg\max_{\boldsymbol{q}\in S^L} p(\boldsymbol{q}|\boldsymbol{x}) = \arg\max_{\boldsymbol{q}\in S^L} \frac{p(\boldsymbol{x}|\boldsymbol{q})p(\boldsymbol{q})}{p(\boldsymbol{x})} = \arg\max_{\boldsymbol{q}\in S^L} p(\boldsymbol{x^1}|\boldsymbol{q^1})p(\boldsymbol{q^1}) \prod_{t=2}^{L} p(\boldsymbol{x^t}|\boldsymbol{q^t})p(\boldsymbol{q^t}|\boldsymbol{q^{t-1}}). \tag{2.43}$$

HMM are mainly limited by the Markov assumption, making them unable to deal with long term dependencies in sequences. Deep learning approaches have made it possible to overcome this problem through the use of Recurrent Neural Networks (RNNs).

2.3.1.2 Recurrent Neural Network

RNNs are specific neural networks which aim at dealing with input signals of variable lengths. They are based on the works of David Rumelhart from 1986 [49]. Let's consider an input signal $\boldsymbol{x} = [\boldsymbol{x^1}, ..., \boldsymbol{x^L}]$, which is made up of L frames $\boldsymbol{x^t}$, with C features per frame ($\boldsymbol{x^t} \in \mathbb{R}^C$). The idea is to process the input sequentially, frame per frame, applying the same weights. The key principle is that the output does not only depend on the input, but also on the previous outputs. It means that the decision process is not local but keeps tracks of what have been output so far ($\boldsymbol{y^1}, ..., \boldsymbol{y^{t-1}}$) and takes it into account for the current output $\boldsymbol{y^t}$. It introduces a time dependency between successive states, as for HMM.

Figure 2.21 presents a visualization of such neural networks. As one can note, $\boldsymbol{y^t}$ is computed based on $\boldsymbol{h^t}$, given that $\boldsymbol{h^t}$ is itself computed based on $\boldsymbol{x^t}$ and $\boldsymbol{h^{t-1}}$. This way, $\boldsymbol{y^t}$ is computed based on all $\boldsymbol{x^i}$ with $i \leq t$. $\boldsymbol{h^t}$ is called the hidden state at time step t; it acts as a memory of past predictions.

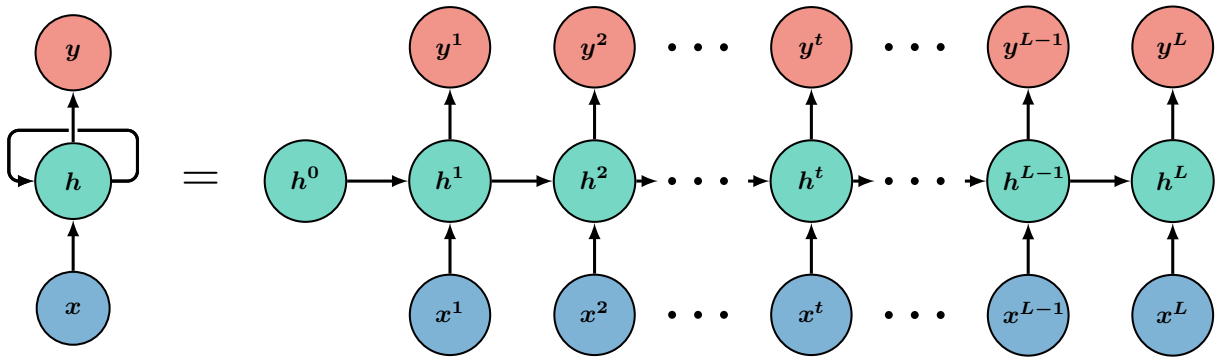


Figure 2.21: Recurrent Neural Network: compressed (left) and unfolded (right) representations.

There are many ways of computing $\boldsymbol{y^t}$ from $\boldsymbol{x^t}$ and $\boldsymbol{h^t}$. Jordan [50] and Elman [51] networks are examples of "Simple Recurrent Networks".

In 1986, M. Jordan proposed the Jordan Network. It follows these equations:

$$\begin{aligned} \boldsymbol{h^t} &= \phi_h(\boldsymbol{W}_x\boldsymbol{x^t} + \boldsymbol{U}_h\boldsymbol{y^{t-1}} + \boldsymbol{b}_h), \\ \boldsymbol{y^t} &= \phi_y(\boldsymbol{W}_y\boldsymbol{h^t} + \boldsymbol{b}_y), \end{aligned} \tag{2.44}$$

where ϕ_h and ϕ_y are activation functions, $\boldsymbol{W}_x$, $\boldsymbol{W}_y$ and $\boldsymbol{U}_h$ are trainable weights of fully-connected layers and $\boldsymbol{b}_h$ and $\boldsymbol{b}_y$ are trainable biases.

The Elman network was proposed in 1990 and is defined as follows:

$$\begin{aligned} \boldsymbol{h^t} &= \phi_h(\boldsymbol{W}_x\boldsymbol{x^t} + \boldsymbol{U}_h\boldsymbol{h^{t-1}} + \boldsymbol{b}_h), \\ \boldsymbol{y^t} &= \phi_y(\boldsymbol{W}_y\boldsymbol{h^t} + \boldsymbol{b}_y). \end{aligned} \tag{2.45}$$

The difference is about the $\boldsymbol{h^t}$ calculation where $\boldsymbol{h^{t-1}}$ is superseded by $\boldsymbol{y^{t-1}}$.

In 1997, the authors of [52] presented the Bidirectional Recurrent Neural Network (BRNN) with the following idea. Some tasks can process whole (offline) data sequences, as opposed to online approaches where the signal is made available over time. It means that one can access to the whole sequence $\boldsymbol{x}$, and instead of just modeling forward unidirectional representations from $\boldsymbol{x^1}$ to $\boldsymbol{x^L}$, the authors suggested combining forward and backward representations simultaneously. This way, the model computes two independent hidden states: $\overrightarrow{\boldsymbol{h^t}}$ for forward pass and $\overleftarrow{\boldsymbol{h^t}}$ for backward pass. $\overrightarrow{\boldsymbol{h^t}}$ and $\overleftarrow{\boldsymbol{h^t}}$ are then concatenated, leading to $\boldsymbol{h^t}$, aggregating knowledge from past and future. This bidirectional strategy can be used whichever the kind of RNN architecture used.

The RNN we have studied so far are limited to one-dimensional sequences. They can be used for tasks from NLP field such as Speech Recognition. RNN have been generalized to multi-dimensional inputs with the Multi-Dimensional Recurrent Neural Network (MDRNN) [53]. MDRNN can be used to handle 2D inputs such as images or 3D inputs like videos. Considering image inputs, $\boldsymbol{h^t}$ would contain context from both top and left pixels for instance. In the same way, the bidirectional concept has been generalized to Multi-Directional RNN for MDRNN. This time, $\boldsymbol{h^t}$ would be the concatenation of 4 passes: from top left, top right, bottom left and bottom right. Increasing the number of dimensions and directions leads to longer and longer computation time, accompanied by a growing number of trainable weights.

RNN are trained using the Backpropagation Through Time algorithm. It is an adapted version of the standard backpropagation we have seen before in which the recurrent layers are unfolded over time, as shown in Figure 2.21. It shows that RNN can be considered as stacks of feed-forward layers. The number of layers in these stacks is dependent to the length of the input, making RNN prone to vanishing and exploding gradient issues. In fact, it induces vanishing gradients on the lowest layers *i. e.* for the first inputs. It means that there is no learning for these first steps, leading to a short memory.

2.3.1.3 Long-Short Term Memory layers

Long-Short Term Memory (LSTM) [54] layers have been proposed in 1997 to tackle the issue of short memory, or in other words, the gradient vanishing problem over time. LSTM integrate internal mechanisms, known as gates, whose aim is to regulate the flow of information.

An LSTM cell is depicted in Figure 2.22. It is made up of 4 gates $\boldsymbol{f^t}$, $\boldsymbol{i^t}$, $\boldsymbol{o^t}$, $\boldsymbol{g^t}$ for forget, input, output and cell gate, respectively. It also introduces a cell state $\boldsymbol{c^t}$, which acts as an additional memory unit. Each gate is computed from the previous hidden state $\boldsymbol{h^{t-1}}$ as

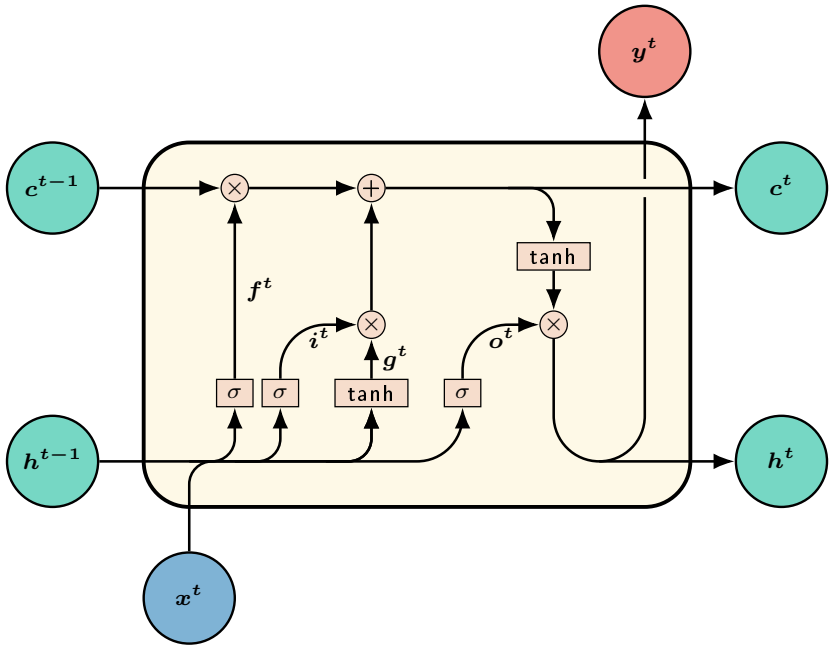


Figure 2.22: Long-Short Term Memory cell.

well as the current input $\boldsymbol{x^t}$, using the sigmoid (or tanh for the cell gate) activation function to output values between 0 and 1, in order to select the relevant information. First, $\boldsymbol{g^t}$ is computed as the new information to integrate to the cell state $\boldsymbol{c^t}$. The forget gate $\boldsymbol{f^t}$ and the input gate $\boldsymbol{i^t}$ quantify the amount of information to keep from the previous cell state $\boldsymbol{c^{t-1}}$ and to integrate from the new input $\boldsymbol{x^t}$, respectively. Finally, the output gate is applied on the new cell state $\boldsymbol{c^t}$ to compute $\boldsymbol{h^t}$ which is also the output $\boldsymbol{y^t}$.

Mathematically, the LSTM cell is defined by:

$$
\begin{aligned}
\boldsymbol{i^t} &= \sigma(\boldsymbol{W}_i\boldsymbol{x^t} + \boldsymbol{U}_i\boldsymbol{h^{t-1}} + \boldsymbol{b}_i),\\
\boldsymbol{f^t} &= \sigma(\boldsymbol{W}_f\boldsymbol{x^t} + \boldsymbol{U}_f\boldsymbol{h^{t-1}} + \boldsymbol{b}_f),\\
\boldsymbol{o^t} &= \sigma(\boldsymbol{W}_o\boldsymbol{x^t} + \boldsymbol{U}_o\boldsymbol{h^{t-1}} + \boldsymbol{b}_o),\\
\boldsymbol{g^t} &= \tanh(\boldsymbol{W}_g\boldsymbol{x^t} + \boldsymbol{U}_g\boldsymbol{h^{t-1}} + \boldsymbol{b}_g),\\
\boldsymbol{c^t} &= \boldsymbol{f^t} \circ \boldsymbol{c^{t-1}} + \boldsymbol{i^t} \circ \boldsymbol{g^t},\\
\boldsymbol{h^t} &= \boldsymbol{o^t} \circ \tanh(\boldsymbol{c^t}),\\
\boldsymbol{y^t} &= \boldsymbol{h^t}.
\end{aligned}
\tag{2.46}
$$

$\boldsymbol{W}_i$, $\boldsymbol{W}_f$, $\boldsymbol{W}_o$, $\boldsymbol{W}_g$, $\boldsymbol{U}_i$, $\boldsymbol{U}_f$, $\boldsymbol{U}_o$, $\boldsymbol{U}_g$ are trainable weights, $\boldsymbol{b}_i$, $\boldsymbol{b}_f$, $\boldsymbol{b}_o$, $\boldsymbol{b}_g$ are trainable biases, σ is the sigmoid activation function and $\circ$ denotes the Hadamard product *i.e .* the element-wise product.

By controlling the information flow, the LSTM solved the long-term memory issue, but it comes at the cost of many additional trainable weights.

2.3.1.4 The Gated Recurrent Unit

In 2014, the authors of [55] proposed the Gated Recurrent Unit (GRU) as an alternative to LSTM, in order to reduce the number of trainable weights, while keeping the long-term memory property. A GRU cell is presented in Figure 2.23. It relies on two gates: an update gate $\boldsymbol{z^t}$ and a reset gate $\boldsymbol{r^t}$.

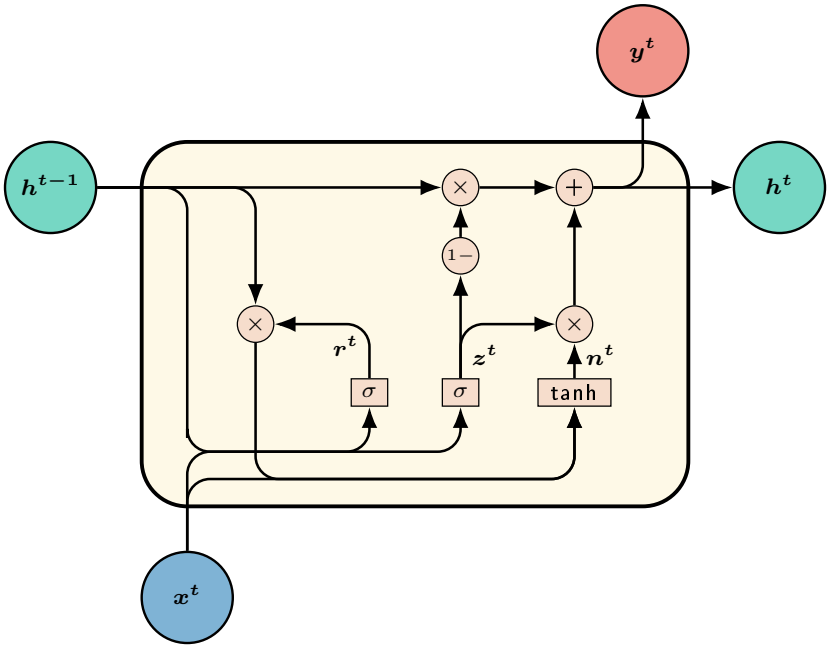


Figure 2.23: Gated Recurrent Unit cell.

Contrary to LSTM, GRU do not rely on a cell state. The gates are also computed on the previous hidden state $\boldsymbol{h^{t-1}}$ and the current input $\boldsymbol{x^t}$. In GRU, the update gate $\boldsymbol{z^t}$ is computed to specify how much to keep from the previous step through $\boldsymbol{h^{t-1}}$ and how much to integrates from the new input through $\boldsymbol{n^t}$. The reset gate $\boldsymbol{r^t}$ also impacts how much information to forget from the past.

GRU cell can be mathematically formulated this way:

$$\begin{aligned}
\boldsymbol{r^t} &= \sigma(\boldsymbol{W}_r\boldsymbol{x^t} + \boldsymbol{U}_r\boldsymbol{h^{t-1}} + \boldsymbol{b}_r),\\
\boldsymbol{z^t} &= \sigma(\boldsymbol{W}_z\boldsymbol{x^t} + \boldsymbol{U}_z\boldsymbol{h^{t-1}} + \boldsymbol{b}_z),\\
\boldsymbol{n^t} &= \tanh(\boldsymbol{W}_n\boldsymbol{x^t} + \boldsymbol{r^t} \circ \boldsymbol{U}_n\boldsymbol{h^{t-1}} + \boldsymbol{b}_n),\\
\boldsymbol{h^t} &= (1 - \boldsymbol{z^t}) \circ \boldsymbol{h^{t-1}} + \boldsymbol{z^t} \circ \boldsymbol{n^t},\\
\boldsymbol{y^t} &= \boldsymbol{h^t}.
\end{aligned} \tag{2.47}$$

$\boldsymbol{W}_r$, $\boldsymbol{W}_z$, $\boldsymbol{W}_n$, $\boldsymbol{U}_r$, $\boldsymbol{U}_z$, $\boldsymbol{U}_n$ are trainable weights, $\boldsymbol{b}_r$, $\boldsymbol{b}_z$, $\boldsymbol{b}_n$ are trainable biases, σ is the sigmoid activation function and $\circ$ denotes the element-wise product.

There is no consensus on the superiority of one of these two cells (GRU and LSTM). It seems that each one enables to reach better results in particular conditions.

We have seen that RNN could be used on multi-dimensional inputs of variable sizes such as images and could combine passes over multiple directions to gather more context. In addition, LSTM and GRU enables to deal with long-term dependencies. However, whether it is HMM or RNN, both approaches lead to output sequences of the same length than that of the input sequences, which made them unsuitable for the HTR task. In the following, we will see how to deal with this issue.

2.3.2 Dealing with inputs and outputs of variable lengths

When dealing with sequence-to-sequence tasks, one faces the problem of the difference between the input sequence length and the target sequence length, which varies from one sample to another. It means that the expected target can be shorter, longer or of the same length than the input sequence. But the point is that one does not know in advance what is the length of the expected target. There are two main approaches to tackle this problem. The first is to predict as many tokens as there are inputs. The challenge is then to find a transformation that enables to adapt the length of the predicted sequence via a post-processing step. The Connectionist Temporal Classification (CTC) follows this approach. The second is to use a recurrent process which outputs the tokens sequentially, step by step, based on the whole input. This recurrent process ends when a specific <end-of-prediction> token is predicted, which enables to have outputs of variable length. This is the idea behind the sequence-to-sequence models.

2.3.2.1 Connectionist Temporal Classification

The CTC [56] was proposed by A. Graves *et al.* in 2006 as an output layer for temporal classification with RNNs. It was first introduced for the tasks of speech and handwriting recognition. The aim was then to solve the alignment problem between the inputs (image or voice recording) and their expected sequence of characters. Indeed, for an image, a character is written on more than a single column of pixels, and for an audio file, the sound corresponding to a phoneme is spread over multiple audio frames.

Let $\boldsymbol{x}$ and $\boldsymbol{y}$ be a couple of input and target output sequences, of lengths L_{x} and L_{y}, respectively, with $L_{\mathrm{x}} \geq L_{\mathrm{y}}$. $\boldsymbol{y}$ is a sequence of tokens $\boldsymbol{y^t} \in \mathcal{A}$, where $\mathcal{A}$ is a given alphabet. The CTC introduces an additional specific null symbol $\varnothing$, also known as blank token, to the original alphabet $\mathcal{A}$: $\mathcal{A}' = \mathcal{A} \cup \{\varnothing\}$. The aim is to determine $p(\boldsymbol{y}|\boldsymbol{x})$.

The CTC consists in a softmax output layer. Let $\boldsymbol{p}$ be the probability lattice ($\boldsymbol{p} \in \mathbb{R}^{L_{\mathrm{x}} \times \mathrm{card}(\mathcal{A}')}$) corresponding to the input $\boldsymbol{x}$, giving probabilities at each frame for each token of the alphabet and for the CTC null symbol.

The CTC involves the use of a many-to-one map β, which enables to map the input sequence with the output sequence: $\beta : \mathcal{A}'^{L} \mapsto \mathcal{A}^{\leq L}$. By many-to-one, we mean that different input sequences can lead to a same output sequence. β consists in a two-step processing: first, the successive identical tokens are removed from the input; then, the null symbols are removed too. This way, the null symbol has two functions: it enables to "predict nothing" and to separate two successive identical tokens in the ground truth. β can be represented

as an automaton defined by a binary transition matrix. Figure 2.24 shows the automaton for the target sequence "CAT". Here, $\beta(\text{CAAAT}) = \beta(\text{CAT}) = \beta(\text{C}\varnothing\text{AAT}) = \text{CAT}$, but $\beta(\text{CCA}\varnothing\text{AT}) = \text{CAAT}$.

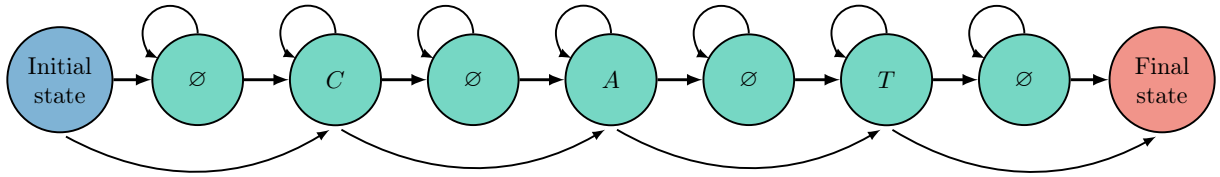


Figure 2.24: CTC automaton for the word "CAT".

$p(\boldsymbol{y}|\boldsymbol{x})$ can then be computed as the sum of all the valid paths $\boldsymbol{\pi} \in \mathcal{A}'^{L_\text{x}}$ leading to $\boldsymbol{y}$ through β:

$$p(\boldsymbol{y}|\boldsymbol{x}) = \sum_{\boldsymbol{\pi} \in \mathcal{B}^{-1}(\boldsymbol{y})} p(\boldsymbol{\pi}|\boldsymbol{x}), \tag{2.48}$$

with

$$p(\boldsymbol{\pi}|\boldsymbol{x}) = \prod_{t=1}^{L_\text{x}} \boldsymbol{p}^{\boldsymbol{t}}_{\boldsymbol{\pi}^{\boldsymbol{t}}}, \forall \boldsymbol{\pi} \in \mathcal{A}'^{L_x}, \tag{2.49}$$

where $\boldsymbol{p}^{\boldsymbol{t}}_{\boldsymbol{\pi}^{\boldsymbol{t}}}$ is the probability of observing label $\boldsymbol{\pi}^{\boldsymbol{t}}$ at position t in the input sequence $\boldsymbol{x}$.

The CTC is used as a loss, dynamically computed through the forward-backward algorithm, similarly to HMM, but by using a binary transition matrix that corresponds to the CTC automaton:

$$\mathcal{L}_\text{CTC}(\boldsymbol{x}, \boldsymbol{y}) = -\ln p(\boldsymbol{y}|\boldsymbol{x}). \tag{2.50}$$

The CTC approach enables to deal with sequence-to-sequence tasks in a straightforward way: all the input tokens are provided at once and β maps it to the final prediction $\hat{\boldsymbol{y}}$. However, there is some important limitations:

- The CTC is only defined for 1D sequences.
- The input length must be greater than or equal to the input length, due to how β is defined.
- The CTC only allows monotonic alignments. This is not a problem for the recognition of text lines but it prevents its use for Neural Machine Translation (NMT) for instance.

We now present the sequence-to-sequence paradigm, which solves the second and third CTC limitations.

2.3.2.2 Sequence-to-sequence paradigm

The seq2seq paradigm was proposed in [57] for the task of NMT. It consists in using a neural network which is made up of two modules: an encoder and a decoder. As represented in Figure 2.25, both encoder and decoder are based on recurrent neural layers. The encoder extracts a representation of the whole input sequence $\boldsymbol{x}$ through the last hidden state $\boldsymbol{e}_{L_\text{x}}$.

This representation, known as context vector $\boldsymbol{c}$, has a fixed size, no matter the input length L_{x}. It enables to handle inputs of variable length. This representation is used to initialize the decoder. The first input of the decoder is a special start-of-prediction token (<sop>). Then, it uses the previous prediction $\boldsymbol{y^{t-1}}$ as input. The prediction is stopped when a special end-of-prediction token (<eop>) is output. This way, the length of the output L_{y} is defined by the prediction of the <eop> token: this is how this paradigm handles the prediction of outputs of variable length.

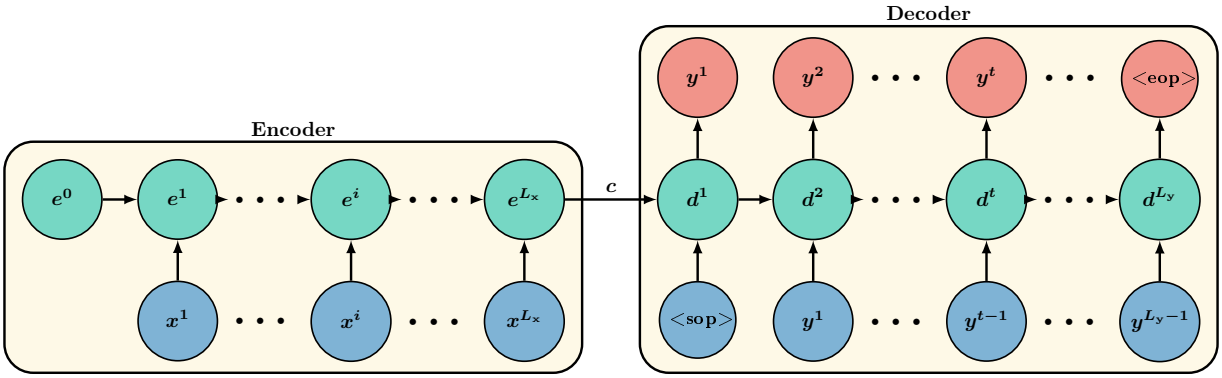


Figure 2.25: Sequence-to-sequence architecture.

More generally, the encoder extracts some features from the input sequence $\boldsymbol{x}$ in the form of a context vector $\boldsymbol{c}$. And the decoder recurrently predicts some tokens based on $\boldsymbol{c}$ and on its own predictions.

The main drawback of this encoder-decoder architecture is the fixed-size representation $\boldsymbol{c}$. No matter the size of the input sequence, $\boldsymbol{c}$ must represent the whole sequence. It leads to poorer results for long sequences. Indeed, the signal cannot be compressed without loss, especially for very long sequences. Intuitively, we can assume that the first elements of the input sequence are under-represented due to the high number of successive gates. The notion of attention mechanism was introduced to tackle this issue.

Attention mechanisms

The idea behind the attention mechanisms is to keep the extracted features of each input frame $(\boldsymbol{e^1}, ..., \boldsymbol{e^{L_{\text{x}}}})$ and to design a way to select the most relevant input frames for the current prediction. Indeed, at each time step t, an attention weight $\boldsymbol{\alpha_i^t}$ is computed for each feature frame $\boldsymbol{e^i}$. Attention weights can be seen as a weighted mask, following a probability distribution: $\sum_{i=1}^{L_{\text{x}}} \boldsymbol{\alpha_i^t} = 1, \forall t$. It is used to compute a weighted sum of the feature frames, leading to a dynamic context vector $\boldsymbol{c^t} = \sum_{i=1}^{L_{\text{x}}} \boldsymbol{\alpha_i^t e^i}$. As shown in Figure 2.26, this attention mechanism enables to get a fixed-size representation at time step t from an input of arbitrary length. $\boldsymbol{c^t}$ can then be used as input for the decoder module. As one can note, the use of an attention mechanism enables to preserve the representation of the whole input sequence.

Attention mechanisms can be classified according to various aspects. First aspect is about the feature frames: attention can be soft, hard or local. Soft attention, as in [58], corresponds to probabilistic attention weights over all the feature frames. The operation is

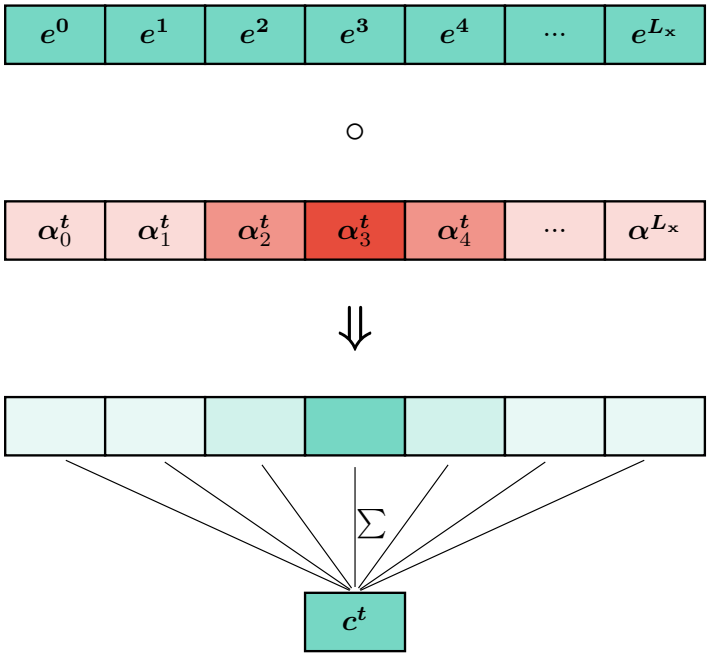


Figure 2.26: Attention mechanism.

differentiable but it can be expensive since it depends on the input size. Hard attention is the opposite, it consists in selecting only one frame (one attention weight to 1 and the others to 0). This way, there is less cost at inference time but the model is non-differentiable. Local attention [59] is an in-between with probabilistic weights over a sub-part of the feature frames. However, it implies some heuristics to choose these sub-parts.

Monotonic attention [60] refers to hard attention in which the feature frames are read only in a given order, from left to right for instance. This way, the frames are read from left to right and processed only if a given score is over a specified threshold. This can only be used for specific tasks implying strict sequentiality. A variant is monotonic chunkwise attention [61], which consists in a monotonic local attention.

The most used attention is the soft attention. It was first proposed in [58] for the task of NMT. This attention mechanism computes an attention score for each feature frame $\boldsymbol{e^i}$ at time step t:

$$\boldsymbol{s_i^t} = \boldsymbol{v}^T \tanh(\boldsymbol{W}\boldsymbol{d^{t-1}} + \boldsymbol{V}\boldsymbol{e^i}), \tag{2.51}$$

where $\boldsymbol{v}$, $\boldsymbol{W}$ and $\boldsymbol{V}$ are weight tensors of fully connected layers. These scores reflect the importance of a given feature frame $\boldsymbol{e^i}$ with respect to the previous hidden state $\boldsymbol{d^{t-1}}$ in deciding the next hidden state $\boldsymbol{d^t}$ and in generating the associated output $\boldsymbol{y^t}$.

Scores are associated with probabilities through a softmax activation, leading to the attention weights:

$$\boldsymbol{\alpha_i^t} = \frac{\exp(\boldsymbol{s_i^t})}{\sum_{k=1}^{L_x} \exp(\boldsymbol{s_k^t})}. \tag{2.52}$$

This attention is known as content-based since the scores are based on the input information $\boldsymbol{e^i}$. In addition, it is qualified as additive attention due to the addition between the two sources of information $\boldsymbol{d^{t-1}}$ and $\boldsymbol{e^i}$ in Equation 2.51.

The next year, Luong *et al.* proposed other ways to compute content-based attention [59]:

- Dot: $\boldsymbol{s_i^t} = \boldsymbol{d^t}^T \boldsymbol{e^i}$.
- General: $\boldsymbol{s_i^t} = \boldsymbol{d^t}^T \boldsymbol{W} \boldsymbol{e^i}$.
- Concat: $\boldsymbol{s_i^t} = \boldsymbol{v}^T \tanh(\boldsymbol{W} \left[\boldsymbol{d^t}; \boldsymbol{e^i}\right])$.

They also proposed a location-based attention *i. e.* the attention weights are only computed from the decoder hidden state:

$$\boldsymbol{\alpha^t} = \text{softmax}(\boldsymbol{W} \boldsymbol{d^t}). \tag{2.53}$$

The authors of [62] proposed to extend the content-based attention from [58] by including some location information in the computation of the attention scores:

$$\boldsymbol{s_i^t} = \boldsymbol{v}^T \tanh(\boldsymbol{W} \boldsymbol{d^{t-1}} + \boldsymbol{V} \boldsymbol{e^i} + \boldsymbol{U} \boldsymbol{f_i^t}), \tag{2.54}$$

with

$$\boldsymbol{f^t} = \boldsymbol{F} \odot \boldsymbol{\alpha^{t-1}}, \tag{2.55}$$

where $\boldsymbol{F}$ is a weight matrix used for a convolution operation. In the following, we will refer to it as hybrid attention since it is both content-aware and location-aware. The authors showed that combining these two information enables a better choice of the feature maps leading to better results in the context of speech recognition.

Whether it is location-based, content-based or hybrid, the attention mechanisms presented so far involve many computations due to the recurrent layers used: each of these attentions is based on $\boldsymbol{d^t}$. It results in long training and prediction times.

Gating mechanism

In 2017, the authors of [63] proposed a gating mechanism for CNN architectures for the language modeling task. It is similar to the gating mechanisms of LSTM or GRU cells. However, instead of controlling the information flow through time, the gating mechanism proposed in [63] focuses on the depth of the network *i. e.* which information must be preserved from one layer to another. This gating mechanism is defined as follows. Given a 2D input representation $\boldsymbol{X} \in \mathbb{R}^{H \times W \times C}$, two projections are computed the same way:

$$\begin{aligned} \boldsymbol{A} &= \boldsymbol{X} \boldsymbol{W} + \boldsymbol{b}, \\ \boldsymbol{B} &= \boldsymbol{X} \boldsymbol{V} + \boldsymbol{c}, \end{aligned} \tag{2.56}$$

with $\boldsymbol{W}$ and $\boldsymbol{V}$ matrices of weights and $\boldsymbol{b}$ and $\boldsymbol{c}$ biases. The output $\boldsymbol{Y}$ is defined as the element-wise product between $\boldsymbol{A}$ and the results of the sigmoid function applied on $\boldsymbol{B}$. This second operand takes its values between 0 and 1, leading to a filtering effect:

$$\boldsymbol{Y} = \boldsymbol{A} \circ \sigma(\boldsymbol{B}). \tag{2.57}$$

This gating mechanism can be seen as a form of self attention because it operates a selection over the input $\boldsymbol{X}$, which is based on $\boldsymbol{X}$ itself. It has also been successfully applied to NMT in [64]. Nowadays, transformers constitute the state-of-the-art model for a majority of tasks that process sequences.

The Transformer architecture

In 2017, Vaswani *et al.* proposed the Transformer architecture [65], depicted in Figure 2.27, for the NMT task. This architecture is made up of N stacked encoder layers followed by N stacked decoder layers. It is based on multi-head, scaled dot-product attention used for self and mutual attention. It also includes embedding, positional encoding and residual components. Contrary to the previous attention-based approaches, the Transformer architecture, although following a recurrent process, does not involve any recurrent layer. It means that the prediction $\hat{\boldsymbol{y}}^{\boldsymbol{t}}$ at timestep t is based on the previous prediction $\hat{\boldsymbol{y}}^{\boldsymbol{t-1}}$. The main difference is that here, $\hat{\boldsymbol{y}}^{\boldsymbol{t}}$ is a raw token, while in the previous presented approaches, the model relied on the decoder hidden state at the previous time step. This difference enables to use teacher forcing during the Transformer training.

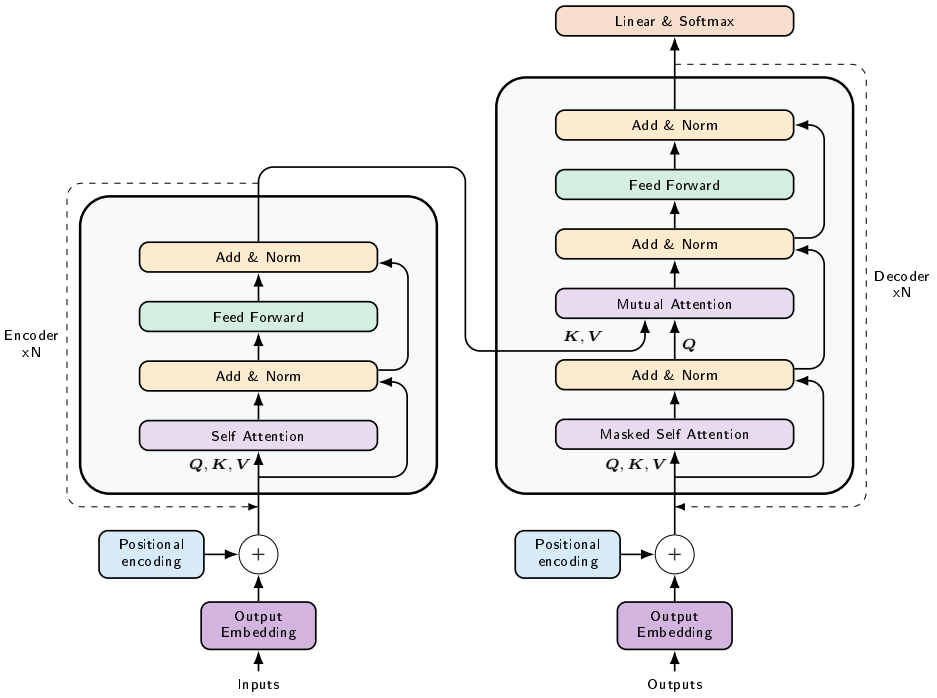


Figure 2.27: Transformer architecture.

Scaled dot-product attention follows the query-key-value paradigm. It is depicted in Figure 2.28. Let's take a set of items $\mathcal{X} = \{\boldsymbol{x}_1, ..., \boldsymbol{x}_L\}$, with $\boldsymbol{x}_i \in \mathbb{R}^{d_{\text{model}}}$. They can be grouped in a matrix $\boldsymbol{X} \in \mathbb{R}^{L \times d_{\text{model}}}$ with $\boldsymbol{X}_i = \boldsymbol{x_i}$. The aim is to retrieve the relevant information among these $\boldsymbol{x_i}$ with respect to a given query representation $\boldsymbol{z} \in \mathbb{R}^{d_{\text{model}}}$. To this aim, we associate to each item $\boldsymbol{x}_i$ a key $\boldsymbol{k}_i \in \mathbb{R}^{d_k}$ and a value $\boldsymbol{v}_i \in \mathbb{R}^{d_v}$ through linear projection. In the same way, we associate a query vector $\boldsymbol{q} \in \mathbb{R}^{d_k}$ to its representation $\boldsymbol{z}$

trough linear projection to fit the dimension of the keys. This can be generalized for a set of M query representations ($\boldsymbol{Z} \in \mathbb{R}^{M \times d_{\text{model}}}$) with matrix computations:

$$\begin{aligned} \boldsymbol{K} &= \boldsymbol{X}\boldsymbol{W}^K, \\ \boldsymbol{V} &= \boldsymbol{X}\boldsymbol{W}^V, \\ \boldsymbol{Q} &= \boldsymbol{Z}\boldsymbol{W}^Q, \end{aligned} \tag{2.58}$$

with $\boldsymbol{W}^K \in \mathbb{R}^{d_{\text{model}} \times d_k}$, $\boldsymbol{W}^V \in \mathbb{R}^{d_{\text{model}} \times d_v}$ and $\boldsymbol{W}^Q \in \mathbb{R}^{d_{\text{model}} \times d_k}$ weight matrices, and $\boldsymbol{K} \in \mathbb{R}^{L \times d_k}$, $\boldsymbol{V} \in \mathbb{R}^{L \times d_v}$ and $\boldsymbol{Q} \in \mathbb{R}^{M \times d_k}$.

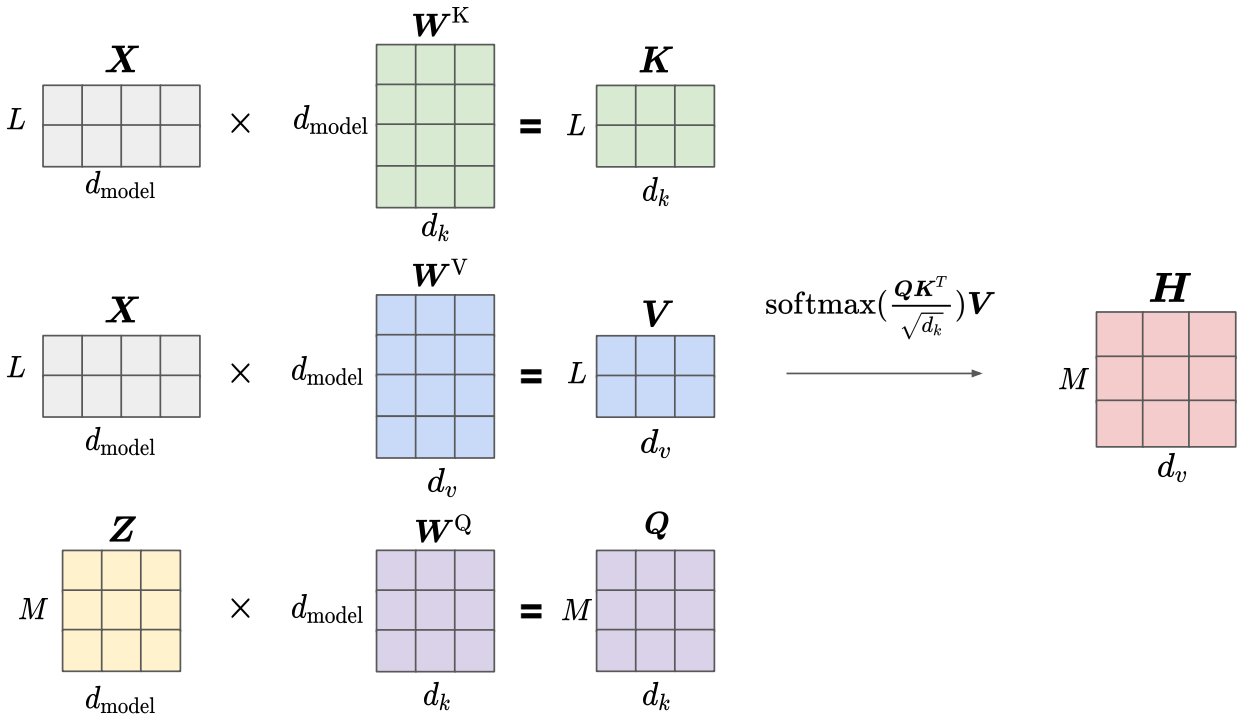


Figure 2.28: Scaled-dot product attention used in the Transformer architecture.

The key $\boldsymbol{k}_i$ is used to compute a matching score with respect to a query $\boldsymbol{q}_j$ through dot product, scaled by a factor $\frac{1}{\sqrt{d_k}}$. The value $\boldsymbol{v}_i$ consists in the relevant information of the item $\boldsymbol{x}_i$ with respect to a given task. This way, given a query $\boldsymbol{q}_j$, a score is computed for each item $\boldsymbol{x}_i$ through their key $\boldsymbol{k}_i$. Probabilities are computed from these scores through softmax and associated to the corresponding values $\boldsymbol{v}_i$ through weighted sum:

$$\boldsymbol{H} = \text{Attention}(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}) = \text{softmax}\left(\frac{\boldsymbol{Q}\boldsymbol{K}^T}{\sqrt{d_k}}\right)\boldsymbol{V}. \tag{2.59}$$

The authors also proposed the concept of multi-head attention, depicted in Figure 2.29. It consists in computing the attention weights h times (once per head $\boldsymbol{H}_i$) in parallel using different projections for the same queries, keys and values. The final context vectors (one per query) $\boldsymbol{C}$ is the projection of the concatenation of these intermediate results:

$$\boldsymbol{C} = \text{MultiHead}(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}) = \text{concat}(\boldsymbol{H}_1, ..., \boldsymbol{H}_h)\boldsymbol{W}^C, \tag{2.60}$$

with

$$\boldsymbol{H}_i = \text{Attention}(\boldsymbol{Z}\boldsymbol{W}_i^Q, \boldsymbol{X}\boldsymbol{W}_i^K, \boldsymbol{X}\boldsymbol{W}_i^V). \tag{2.61}$$

$\boldsymbol{W}_i^K \in \mathbb{R}^{d_{\text{model}} \times d_k}$, $\boldsymbol{W}_i^V \in \mathbb{R}^{d_{\text{model}} \times d_v}$, $\boldsymbol{W}_i^Q \in \mathbb{R}^{d_{\text{model}} \times d_k}$ and $\boldsymbol{W}^C \in \mathbb{R}^{h \cdot d_v \times d_c}$ are matrices of weights. The authors set $d_v = d_k = d_{\text{model}}/h$ and $d_c = d_{\text{model}}$. Multi-head attention aims at combining information from different positions and in different sub-spaces to improve the expressiveness of the context vector.

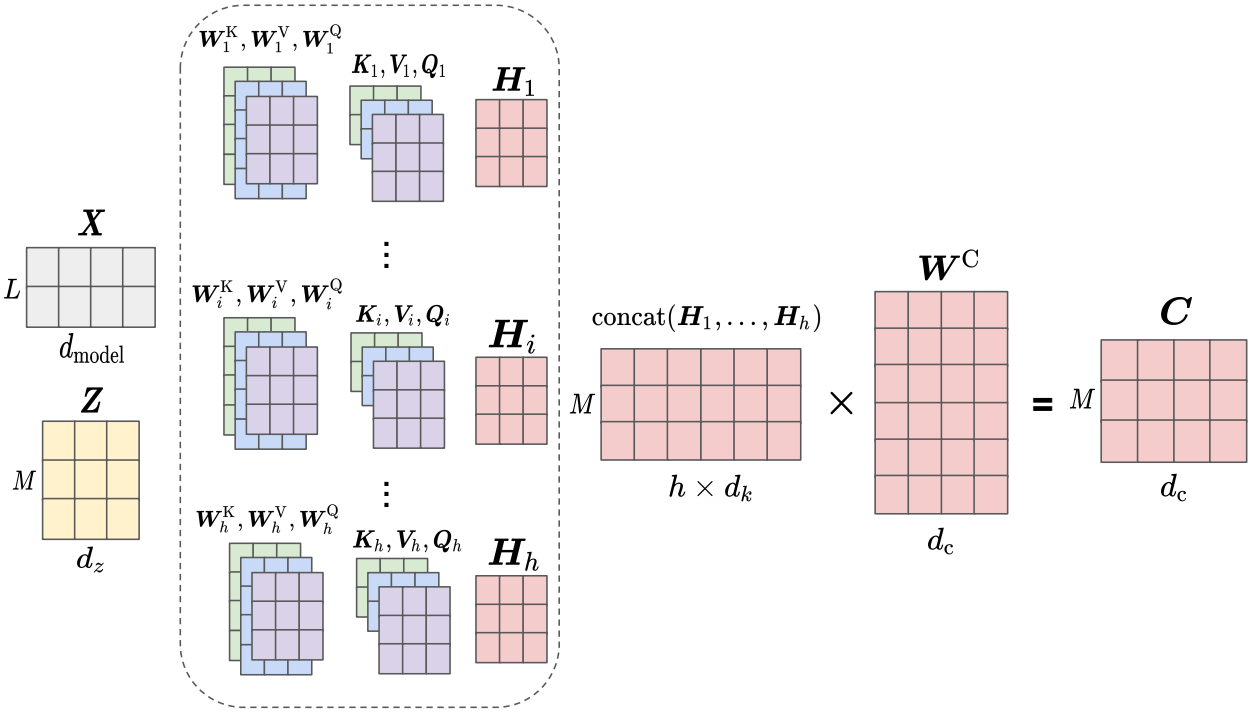


Figure 2.29: Multi-head scaled-dot product attention used in the Transformer architecture.

Based on these definitions, self attention refers to multi-head scaled dot-product applied with queries, keys and values from a same source sequence. In the encoder, the source sequence corresponds to the input. In the decoder, the source sequence is the previous predicted tokens. Mutual attention is also a multi-head scaled dot-product attention, but the queries come from a target sequence, different from the source sequence, from which the keys and the values are computed. The target sequence is the already predicted tokens and the source sequence is the output of the encoder *i. e.* a representation of the input.

As mentioned previously, the Transformer architecture does not rely on any recurrent layer. This way, the queries cannot be based on a hidden state of a recurrent layer; it is directly based on raw predicted tokens. One can then use teacher forcing to parallelize the computations at training time. This way, a mask is used to prevent using future predictions to compute the current one. Since inputs and outputs are raw tokens, some embedding layers are used to increase the dimension of the model.

In addition to removing the recurrent layers, the Transformer architecture does not rely on any convolutional component either. Dependencies are only modelized through the attention layers, by weighted sums of the sequence frames. However, there is no notion of sequence:

frames are considered as a set, not as a sequence. To tackle this problem, the authors proposed to add positional encoding to the embedding in order to inject information about this sequentiality. The positional encoding used in this work corresponds to sine and cosine functions of different frequencies:

$$\begin{aligned} &\mathrm{PE}(j, 2k) = \sin(w_k \cdot j), \\ &\mathrm{PE}(j, 2k+1) = \cos(w_k \cdot j), \\ &\forall k \in [0, d_{\text{model}}/2], \end{aligned} \tag{2.62}$$

with

$$w_k = 1/10000^{2k/d_{\text{model}}}, \tag{2.63}$$

where j is the position and k is the dimension. Computing the positional encoding in this way bring many advantageous properties:

- It is fixed and deterministic: it does not imply trainable weights.
- Positional encoding does not depend of the total length of the sequence.
- Encoding values are bounded and can generalize to unseen lengths thanks to the sine and cosine functions.
- The encoding is unique for each position thanks to the combination of frequencies.

The Transformer architecture has exceeded the state of the art in many areas. However, there is still room for improvement. Indeed, the self-attention layer is a bottleneck when dealing with very long sequences: it requires quadratic computation time to produce the attention scores. In addition, the recurrent process combined with the stacked layers leads to inference time that could not satisfy most of industry applications. Many works attempted to improve the Transformer architecture to alleviate these issues: [66, 67, 68, 69].

The Transformer architecture, and more generally the attention-based and sequence-to-sequence architectures, are designed to deal with one-dimensional input sequences. As a result, standard sequence-to-sequence approaches cannot be used as is for HTR. On the other hand, we have seen that powerful computer vision architectures (CNN, FCN) were proposed to handle input images. This way, it would be better to consider HTR as an in-between: an image-to-sequence (or document-to-sequence) problem, which is closer to reality.

2.4 HTR: an image-to-sequence problem

On the one hand, the emergence of CNN enabled to process images more efficiently, modeling spatially dependent information, also bringing translation invariance and weight sharing. One can also reduce the number of trainable weights by using Depthwise Separable Convolution. In addition, one can deal with inputs of variable sizes using FCN. On the other hand, recurrent layers and sequence-to-sequence architectures enabled to deal with one-dimensional input sequences and one-dimensional output sequences of variable lengths.

HTR, as well as image captioning or scene text recognition, are part of the tasks that lie between computer vision and sequence-to-sequence problems. The evolution of the architectures dedicated to these tasks has followed those of these two fields.

For instance, the image captioning task was first handled by a hybrid architecture mixing CNN and LSTM as in [70]. A CNN encoder is used to extract features from the input image, generating a vector of fixed size. The LSTM layer is then used to predict the caption, word by word, based on this feature vector.

The authors of [71] proposed to introduce an attention mechanism in this kind of models. By doing this, one can use an FCN in order to avoid collapsing the horizontal and vertical axes of the image for the extracted features. A flatten operation enables to reshape the features into a one-dimensional sequence. A self content-based attention mechanism, combined with an LSTM layer, is used to select the feature frames to focus on, while recurrently predicting the caption.

CNN was then combined with a transformer decoder in [72] for HTR. 1D positional encoding is superseded by 2D positional encoding for the flattened features to preserve the 2D nature of the image. It means that the position is still encoded using sine and cosine functions, but half of the dimensions are dedicated to the horizontal position and the other half to the vertical position.

Whether it is CNN+LSTM, CNN+LSTM with soft attention or CNN+transformer, these architectures only represent a concatenation of an encoder sub-network coming from computer vision for feature extraction, and a decoder sub-network from the sequence-to-sequence paradigm for sequence prediction. The connection of these two sub-networks is made possible by some tricks: the flatten operation, the 2D positional encoding.

More recently, the authors of [73] proposed the Vision Transformer (ViT) for the task of image captioning. It consists in applying the transformer encoder on the raw image pixels, without using any convolutional component. To this end, they apply a pre-processing stage to the input image: the image is split into fixed-size patches and flattened, before being linearly projected. An MLP is used after the transformer encoder to predict the class probabilities. This concept has also been used in [74] for image classification and adapted to image-to-sequence tasks by also using the transformer decoder part, such as in [75] for image captioning or in [76] for scene text recognition.

Vision Transformer provides similar or even better results compared to CNN-based approaches. It could be due to the difficulty of CNN to model very long dependencies which depends on the size of the receptive field. On the contrary, transformer-based approaches rely on self attention which consists in a soft attention over the whole input sequence: dependencies between first and final items can theoretically be modeled as of the first encoder layer.

We can therefore ask ourselves if the CNN are still relevant for image-to-sequence problems. From these first works, it seems that ViT are prevalent only for very large datasets. Indeed, CNN would perform better with few data compared to ViT because it encodes prior knowledge about the image domain like translation invariance and spatial dependencies; ViT must learn it.

We have formulated the HTR task as an image-to-sequence problem and we have seen the major issues related to the use of deep neural networks to handle such a complex task. We will now focus on the evaluation of the performance of such systems.

2.5 Evaluation environment for HTR

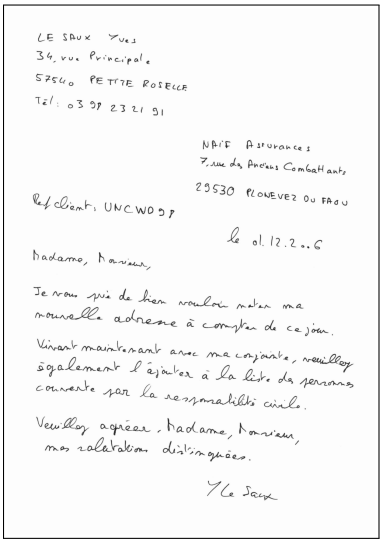
To compare objectively the performance of different neural networks, one must evaluate the different approaches in the same conditions. This way, we evaluated our proposed models on public datasets and using the same metrics.

2.5.1 Datasets

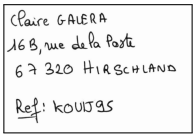
There are very few available public datasets for the HTR task. To evaluate the approaches we proposed, we used the three following public datasets: RIMES, IAM and READ. Although these datasets provide images of whole pages, they are generally annotated for line-level approaches, as we will see in the next chapter.

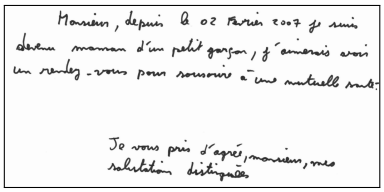
2.5.1.1 RIMES

RIMES (Recognition and Indexing of handwritten documents and faxes) is a project funded by the French ministries of defense and research. Its goal was to create a new dataset of mixed (handwritten and typed) documents, dedicated to varied tasks representative of industrial applications. The idea was to cover the following tasks: layout analysis, handwriting recognition, information extraction, writer identification and logo identification.



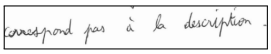
(a) RIMES 2009 - page.

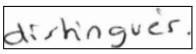




(b) RIMES 2009 - paragraph.

(c) RIMES 2011 - paragraph.





(d) RIMES 2011 - line.

Figure 2.30: RIMES dataset examples.

The RIMES dataset corresponds to 12,723 pages collected from volunteers in the form of 5,605 handwritten mails of two to three pages, whose content corresponds to a fictional correspondence between an individual and a company. Each mail is composed of a handwritten letter page, a form page and an optional fax cover sheet page. The volunteers could write up to 5 mails, each one following one of the 9 proposed scenarii, such as change of personal

information or modification of contract. These mails have been scanned at a resolution of 300 DPI, with a gray-scale encoding.

It led to two main evaluation campaigns and a competition, providing two sub-datasets from this RIMES dataset in 2009 and 2011, respectively. We named them RIMES 2009 and RIMES 2011.

RIMES 2009

RIMES 2009 corresponds to the dataset used in the second evaluation campaign [4]. It is made up of 1,050 pages for training, 100 pages for validation and 100 pages for test. All the pages are the first of a subset of the mails *i. e.* single-page handwritten letters. RIMES 2009 was proposed for letter layout analysis, handwriting recognition and writer identification. Text regions are localized and classified into 8 categories: sender, recipient, date & location, subject, opening, body, signature and PS & attachment. This way, for a given document image, there are many sequences of characters which represent the different text regions, each one associated to a given class. Bounding boxes are also provided for each text region; this enables to also process the dataset for paragraph-level HTR.

It has to be noted that this version of the dataset was only made available to participants of the evaluation campaign. At that time, the recognition tasks only consisted in recognizing isolated characters or words. To our knowledge, we are the first to use it at paragraph and page levels.

RIMES 2011

RIMES 2011 corresponds to the dataset provided in the handwriting competition at ICDAR 2011 [1]. Images are bodies of a subset of the handwritten letters. RIMES 2011 dataset also provides annotations of bounding boxes of the text lines, leading to a line-level version of RIMES 2011. This way, the images of bodies are split into text line images and associated to the corresponding ground truth transcription.

Samples of the different versions of the RIMES dataset are shown in Figure 2.30. Table 2.3 provides statistical information about the different RIMES datasets. As one can note, samples from RIMES 2009 contain up to 2,719 characters at page level, with a mean of 578 characters. One can also notice that the paragraphs from RIMES 2009 are more diverse in terms of image size and text length. This is due to the fact that paragraphs from RIMES 2011 only correspond to bodies of letters whereas paragraphs from RIMES 2009 can be of any class (body, recipient or sender coordinates, or subject for instance).

In Table 2.4, we specified the split used in terms of training, validation and test, as well as the charset size and the total number of characters in the dataset. For RIMES 2011 at line level, there are two splits used in the literature: line-1, which is the line version of the RIMES 2011 dataset at paragraph level; and line-2 which is the mainly used split by the community. For line-2, text lines from a same paragraph can be found in both training and validation sets for example. This way, line-1 is to be preferred when pre-training on line images before training on paragraph images. The difference of characters from page to paragraph or from paragraph to lines can be explained by the addition of some line breaks. It can also be due to annotation errors.

Table 2.3: RIMES datasets statistics.

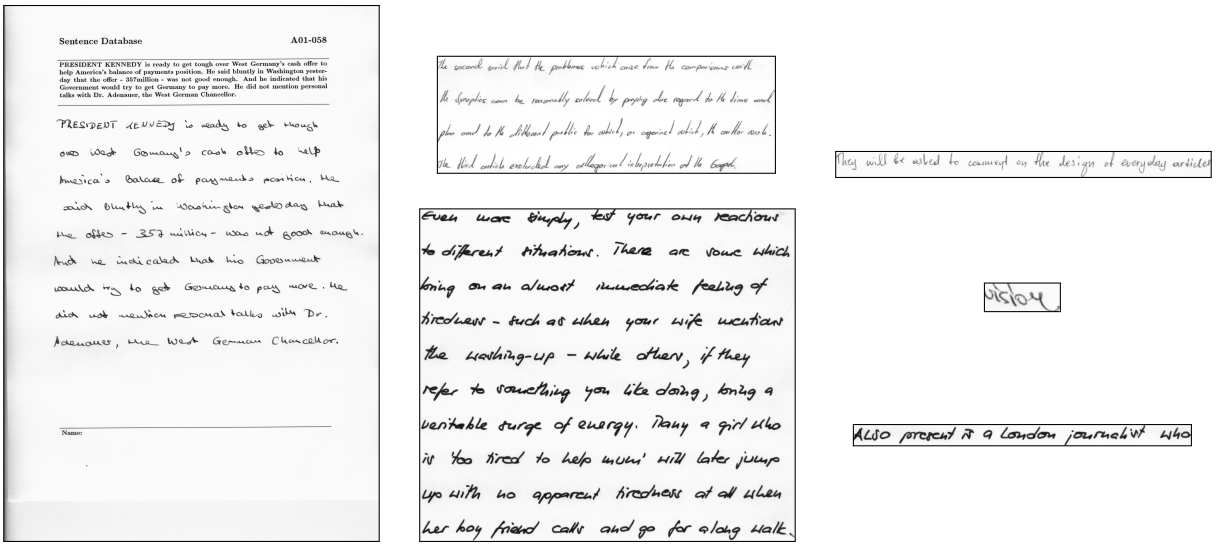
	Min.	Q1	Median	Q3	Max.	Mean
RIMES 2009 - page						
Width (px)	2,446	2,466	2,470	2,472	2,480	2,469
Height (px)	3,464	3,500	3,502	3,504	3,512	3,502
Lines	5	16	18	20	43	18
Words	36	93	113	138	558	120
Chars	157	454	550	660	2,719	578
RIMES 2009 - paragraph						
Width (px)	112	738	928	1,297	2,464	1,123
Height (px)	64	150	298	514	2,276	431
Lines	1	1	3	4	24	3
Words	1	5	9	20	414	21
Chars	3	21	47	79	2,043	103
RIMES 2011 - paragraph						
Width (px)	1,464	2,108	2,228	2,336	2,468	2,204
Height (px)	346	846	1,061	1,282	2,132	1,081
Lines	2	6	7	9	18	8
Words	12	48	65	85	245	70
Chars	71	244	322	422	1,182	348
RIMES 2011 - line						
Width (px)	96	1,346	1,828	2,044	2,462	1,636
Height (px)	34	104	126	152	364	129
Words	1	7	9	12	24	9
Chars	2	34	47	57	110	45

Table 2.4: RIMES datasets split.

	Training	Validation	Test	charset size	# chars
RIMES 2009 - page	1,050	100	100	108	723k
RIMES 2009 - paragraph	5,875	540	559	108	717k
RIMES 2011 - paragraph	1,400	100	100	98	556k
RIMES 2011 - line-1	10,530	801	778	97	546k
RIMES 2011 - line-2	9,947	1,333	778	97	543k

2.5.1.2 IAM

The IAM dataset was first published in 1999 in the International Conference on Document Analysis and Recognition (ICDAR) [77], and updated in [2]. We used the third version of this dataset. It consists in 1,539 form pages of handwritten English text scanned at a resolution of 300 DPI. Scans are saved as PNG images with 256 gray levels. The writers, 657 in number, were asked to copy the printed text at the top of a form. The printed texts correspond to fragments of text from the Lancaster-Oslo/Bergen (LOB) corpus [78], a collection of 500 English texts.



(a) IAM - Original form. (b) IAM - paragraph. (c) IAM - line.

Figure 2.31: IAM dataset examples.

The IAM dataset includes ground truth transcription as well as segmentation annotation at several levels: paragraph (body), line and word. We only used the paragraph and line levels.

Samples of the IAM dataset are shown in Figure 2.31. We did not use the original forms from the IAM dataset because the printed text part consists in the exact same text than the handwriting part. Indeed, this would have biased the training. As one can note, the writing style can differ a lot from one writer to another, comparing the two paragraph images.

Table 2.5: IAM datasets statistics.

	Min.	Q1	Median	Q3	Max.	Mean
Paragraph						
Width (px)	1,574	1,816	1,874	1,950	2,266	1,891
Height (px)	356	1,253	1,522	1,730	2,102	1,488
Lines	2	7	9	10	13	9
Words	13	70	78	87	150	79
Chars	52	340	379	420	634	382
Line						
Width (px)	100	1,672	1,758	1,834	2,260	1,699
Height (px)	38	100	120	144	342	124
Words	1	7	9	11	53	9
Chars	1	38	43	49	93	43

Statistics about IAM datasets are given in Table 2.5. Looking at both paragraph and line levels, one can notice that the IAM dataset follows the same trend than the RIMES 2011 dataset, with an increasing size variability from paragraphs to lines. IAM paragraphs are made up of 2 to 13 lines, with a mean of 9 lines. It is a bit less than for paragraphs of

RIMES 2011. The number of characters per line is, on average, nearly identical compared to the RIMES 2011 dataset with 43 characters compared to 45.

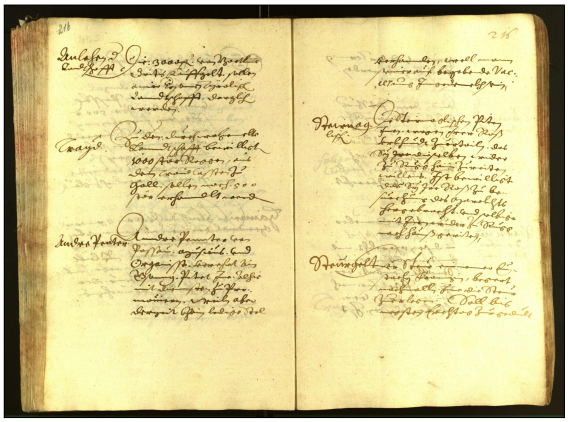
We used the Aachen split [79] for both paragraph and line datasets, as detailed in Table 2.6.

Table 2.6: IAM datasets split.

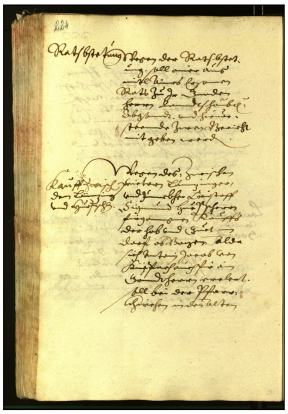
	Training	Validation	Test	charset size	# chars
Paragraph	747	116	336	80	458k
Line	6,482	976	2,915	79	449k

2.5.1.3 READ

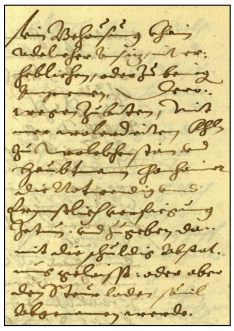
READ (Recognition and Enrichment of Archival Documents) is a European project whose mission was to improve the access to archival documents through the use of HTR and keyword spotting technologies. One of the READ datasets was proposed in a competition of the International Conference on Frontiers in Handwriting Recognition (ICFHR) in 2016 [3]. We named it READ 2016.



(a) READ-double-page.



(b) READ-page.



(c) READ-paragraph.

(d) READ-line.

Figure 2.32: READ 2016 dataset examples.

The READ 2016 dataset is made up of 450 pages of Early Modern German. It consists of a subset of documents from the Ratsprotokolle collection, which is composed of about 30,000 pages of minutes of the council meetings, dated from 1470 to 1805. The number of writers is unknown. Contrary to RIMES or IAM, it is encoded as RGB images.

This dataset is originally at single-page level. We generated a double-page version of this dataset by merging successive pages. Only very few pages were not paired: we removed them from the dataset. This dataset also provides, in addition to the text annotation, segmentation labels at paragraph and line levels. We used them for the HTR task at these levels.

Samples of the READ 2016 datasets are shown in Figure 2.32. As one can notice, handwriting from READ 2016 are far more difficult to read than those of RIMES and IAM. This is mainly due to the background, noised by the bleed-through effect, combined with the historical way of writing and the language.

Statistics about this dataset are provided in Table 2.7. As for RIMES 2009 at page level, samples from READ 2016 at single-page and double-page levels are close in terms of size. One should note that the paragraphs also integrate images of page number leading to very small images of a single character. It results in a huge variety in the text length going from 1 to 758 characters.

Table 2.7: READ 2016 datasets statistics.

	Min.	Q1	Median	Q3	Max.	Mean
Double-page						
Width (px)	4,714	4,770	4,786	4,802	4,874	4,793
Height (px)	3,494	3,510	3,510	3,534	3,630	3,523
Lines	18	45	48	50	59	47
Words	48	177	196	218	276	194
Chars	296	963	1,072	1,204	1,455	1,061
Single-page						
Width (px)	2,168	2,346	2,394	2,450	2,562	2,398
Height (px)	3,494	3,510	3,510	3,534	3,630	3,523
Lines	6	22	24	25	31	23
Words	18	86	98	111	153	97
Chars	95	466	536	604	807	528
Paragraph						
Width (px)	64	430	716	1,340	1,992	863
Height (px)	34	176	352	910	3,000	636
Lines	1	1	3	8	26	5
Words	1	2	8	33	143	22
Chars	1	7	39	186	758	119
Line						
Width (px)	42	930	1,056	1,164	1,626	962
Height (px)	22	100	114	132	514	121
Words	1	3	4	5	13	4
Chars	1	19	23	27	43	22

The different splits used in this thesis are detailed in Table 2.8. One can note that there are fewer examples at single-page and double-page levels than in RIMES 2009 at page level. However, for the line-level version, it is comparable to RIMES and IAM.

Table 2.8: READ 2016 datasets split.

	Training	Validation	Test	charset size	# chars
Double-page	169	24	24	89	230k
Single-page	350	50	50	89	238k
Paragraph	1,602	182	199	89	236k
Line	8,367	1,043	1,140	88	227k

2.5.2 Metrics

The HTR task is generally handled in three steps, including a segmentation stage and a recognition stage. The segmentation part aims at detecting the text regions in the input images, and the recognition part aims at recognizing the sequence of characters inside these different text regions. We now go over the main metrics used in both segmentation and recognition stages.

2.5.2.1 Segmentation metrics

Segmentation predictions and annotations are generally bounding boxes representing text regions. The main difficulty to compute the metrics is that the number of predictions may not match the actual number of text regions in the ground truths: the model can predict 3 text lines whereas there are 4 text lines in the image, for instance. Two metrics are generally used to evaluate the performance of the segmentation stage: the Intersection over Union (IoU) and the mean Average Precision (mAP).

IoU

The Intersection over Union is defined as the area of the intersection between the ground truth $\boldsymbol{Y}$ and the prediction $\hat{\boldsymbol{Y}}$, divided by the area of the union of these two areas:

$$\text{IoU} = \frac{\boldsymbol{Y} \cap \hat{\boldsymbol{Y}}}{\boldsymbol{Y} \cup \hat{\boldsymbol{Y}}}. \tag{2.64}$$

Calculated in this way, the IoU enables to measure the quality of the segmentation process. On the one hand, it ensures that the prediction encompasses the annotation as much as possible, via the intersection. And on the other hand, it avoids the over-segmentation thanks to the division by the union, preventing to predict the whole document as a text region.

In the case of multiple text regions in a same input image, the IoU can be averaged between the different predictions. Predictions are ordered by their confidence score and associated to the ground truth which leads to the highest IoU.

mAP

An alternative metric to evaluate the segmentation stage is the mean Average Precision [80, 81]. It corresponds to a mean of the Average Precision (AP) of the different classes to be recognized. AP is computed as the area under the precision-recall curve. Precision and recall are metrics based on ratios of True Positive (TP), False Positive (FP) and False Negative (FN). The precision corresponds to the ratio of correct predictions over all the predictions:

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}, \tag{2.65}$$

while the recall measures the ratio of predictions that match the annotation:

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}. \tag{2.66}$$

Rank	TP/FP	Precision	Recall	p_{inter}
1	TP	1/1	1/4	1
2	FP	1/2	1/4	1
3	TP	2/3	2/4	3/4
4	TP	3/4	3/4	3/4
5	FP	3/5	3/4	3/4
6	TP	4/6	4/4	4/6
7	FP	4/7	4/4	4/6

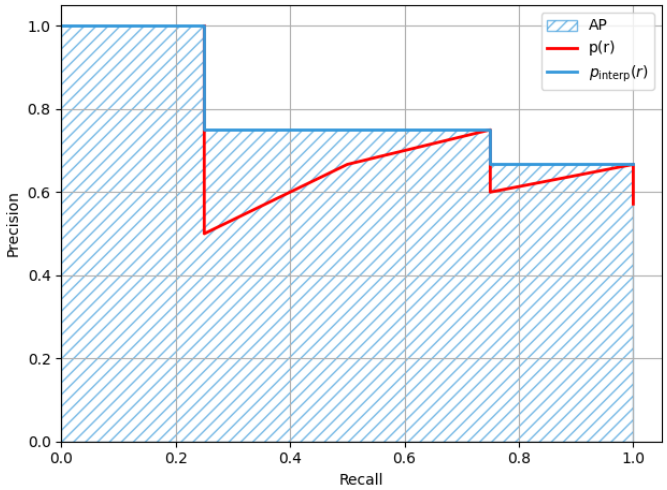


Figure 2.33: AP computation example. Left: ordered predictions. Right: associated Precision/Recall curve.

A prediction is considered as true positive with respect to a bounding box annotation if the IoU between them is higher than a given threshold. This way, one can compute the AP for a given class and a specific IoU threshold as follows:

- The predictions are ordered by their confidence score.
- Precision $p(r_n)$ and recall r_n are computed following this order, for each prediction.
- For k predictions, the Average Precision (AP) is computed as the interpolation of the area under the precision/recall curve, considering $r_0 = 0$:

$$\text{AP} = \sum_{n=0}^{k} (r_{n+1} - r_n) \cdot p_{\text{interp}}(r_{n+1}), \tag{2.67}$$

with

$$p_{\text{interp}}(r_{n+1}) = \max_{\tilde{r} > r_{n+1}} p(\tilde{r}). \tag{2.68}$$

Figure 2.33 provides an example of AP computation for 7 predictions and 4 ground truth annotations. It leads to:

$$\text{AP} = (\frac{1}{4} - 0) \times 1 + (\frac{2}{4} - \frac{1}{4}) \times \frac{3}{4} + (\frac{3}{4} - \frac{2}{4}) \times \frac{3}{4} + (\frac{4}{4} - \frac{3}{4}) \times \frac{4}{6} \approx 79\% \tag{2.69}$$

The mAP can then be computed as the mean of the AP over a set of classes $\mathcal{C}$:

$$\text{mAP} = \frac{1}{\text{card}(\mathcal{C})} \sum_{c \in \mathcal{C}} \text{AP}_c. \tag{2.70}$$

There are various ways of computing the mAP. A common way is to weight the AP values by the number of items per class or by the number of pixels involved in each class. In addition, the mAP is generally computed for several IoU thresholds so as to to take their average. Standard IoU thresholds are from 50% to 95% with a step of 5%.

Both IoU and mAP are debatable. More generally, designing a metric to evaluate a segmentation task is very complex in nature. Indeed, the annotation of the different elements to be segmented is subject to variations due to the subjectivity of the annotators. Thus, the main metrics to rely on are the ones that evaluate the target task *i. e.* the recognition metrics.

2.5.2.2 Recognition metrics

To evaluate the accuracy of the text recognition, two main metrics are computed: the Character Error Rate (CER) and the Word Error Rate (WER). Both metrics are based on the Levenshtein distance. It is defined between two sequences of symbols $\boldsymbol{s}_\text{A}$ and $\boldsymbol{s}_\text{B}$ as the minimal cost to transform $\boldsymbol{s}_\text{A}$ into $\boldsymbol{s}_\text{B}$ while only using the following elementary operations: insertion, deletion and substitution. Edit cost is set to one for each operation. The Levenshtein distance is computed through dynamic programming as follows:

$$\text{lev}(\boldsymbol{s}_\text{A}, \boldsymbol{s}_\text{B}) = \begin{cases} \max(|\boldsymbol{s}_\text{A}|, |\boldsymbol{s}_\text{B}|) & \text{if } \min(|\boldsymbol{s}_\text{A}|, |\boldsymbol{s}_\text{B}|) = 0, \\ \text{lev}(\boldsymbol{s}_{\text{A}_{[1:]}}, \boldsymbol{s}_{\text{B}_{[1:]}}) & \text{if } \boldsymbol{s}_{\text{A}_0} = \boldsymbol{s}_{\text{B}_0} \\ 1 + \min \begin{cases} \text{lev}(\boldsymbol{s}_{\text{A}_{[1:]}}, \boldsymbol{s}_\text{B}) \\ \text{lev}(\boldsymbol{s}_\text{A}, \boldsymbol{s}_{\text{B}_{[1:]}}) \\ \text{lev}(\boldsymbol{s}_{\text{A}_{[1:]}}, \boldsymbol{s}_{\text{B}_{[1:]}}) \end{cases} & \text{otherwise.} \end{cases} \tag{2.71}$$

In this equation, $\boldsymbol{s}_{\text{B}_{[1:]}}$ denotes the sequence $\boldsymbol{s}_\text{B}$ in which the first item is removed. $|\boldsymbol{s}_\text{A}|$ is the length (cardinality) of the sequence $\boldsymbol{s}_\text{A}$.

CER is defined as the levenshtein distance between two strings: the ground truth text $\boldsymbol{y}$ and the predicted text $\hat{\boldsymbol{y}}$. They are processed as sequences of characters, normalized by the length of the ground truth. The global CER for a set of K strings is computed as follows:

$$\text{CER} = \frac{\sum_{i=1}^{K} \text{lev}(\hat{\boldsymbol{y}}_i, \boldsymbol{y}_i)}{\sum_{i=1}^{K} |\boldsymbol{y}_i|}, \tag{2.72}$$

WER follows the exact same formula but strings are processed as sequences of words. If not stated otherwise, punctuation characters are considered as words, as in [3].

2.6 Conclusion

In this chapter, we formalized the HTR task as an image-to-sequence problem. We provided fundamental background on deep learning approaches and presented the main components and techniques proposed in the literature to deal with common issues: training, generalization, convergence and vanishing and exploding gradients. We particularly focused on computer vision and sequence-to-sequence fields of research which are the two pillars of image-to-sequence problems. This way, we covered the main neuronal components and how to train them. We also presented the evaluation part, including datasets and metrics.

At the beginning of this work, line-level approaches dominated the state of the art (this is still the mainly used approach currently). In fact, to our knowledge, the only works proposing to recognize multiple lines of text *i. e.* whole paragraphs, in an end-to-end way, without any explicit line segmentation stage, were only two [82, 83]. Our ultimate goal was, from the early beginning, to be able to recognize the text of whole documents, without any explicit segmentation step, in the hope of further improving the performance. To this end, we decided to go step by step, from single text line recognition to paragraph recognition, and then to the recognition of whole documents. In addition to propose end-to-end architectures for HTR at line, paragraph and document levels, the following constraints were also of primary consideration with the aim of designing modules as generic as possible:

- The architecture must be able to deal with input images of variable sizes to preserve the height/width ratio so as to avoid deformations and information losses. In addition, it enables transfer learning between line-level architectures and paragraph-level or document-level architectures.

- The model must reach at least competitive results. To this aim, and to be comparable with state-of-the-art approaches, the model must be trained without using any pre-trained model on external data, and without being trained itself on external data. In addition, to really compare the performances of raw architectures, they must be evaluated without using an external language model.

- The number of trainable parameters, the training time as well as the prediction time must be reasonable, to respect hardware limitations.

- The model must be trained with few training data. This is a limitation due to the public datasets we used. In addition, since the annotation cost is essential, we aim at using as few annotations as possible.

Based on the image-to-sequence study we provided in this chapter, and on these objectives, we decided to design a generic FCN encoder for the feature extraction part from the input image. Indeed, FCN involves a reasonable number of parameters, operations are parallelizable, inputs can be of variable sizes, and it requires fewer labeled data compared to transformer-based encoder for instance, thanks to its translation invariance and spatial

dependencies properties. This way, an FCN encoder could be used by any system dedicated to the HTR task, it is especially useful for transfer learning purposes. On the other hand, we focused on the attention mechanisms, in particular the transformer architecture, for the decoder part. As a matter of fact, the soft attention enables to preserve the whole encoded representation, without compression through decoding. In addition, it also enables to model dependencies across the whole signal and to focus on specific part of the input signal: this seemed to be very relevant when dealing with reading orders. However, as we have seen, designing and training deep and complex architectures raises several issues (overfitting, convergence) that we must handle.

We will now focus on the contributions of this thesis in details. The three following chapters are dedicated to line-level, paragraph-level and document-level recognition, respectively. For each one, we introduce the context and the challenges and we present the related works and the contributions.

Chapter 3

Handwritten text line recognition

Historically, Handwritten Text Recognition (HTR) was performed at character level by splitting the task into four main steps [84]: pre-processing, segmentation, representation and Optical Character Recognition (OCR) [85, 86, 87, 88]. Many pre-processings were used to make the segmentation and the recognition easier. For example, binarization, noise reduction, skew correction or slant removal are common pre-processing that were used. Segmentation was performed to extract isolated images of characters. This character segmentation was mainly based on heuristics, such as inter-line and inter-word spacing, to first segment the document into lines, then words, and finally characters. The representation step consisted in detecting features [89] through transformation or statistical approaches. The character recognition part was mainly carried out by template matching approaches, statistical methods such as Hidden Markov Model (HMM) or clustering analysis, and more recently neural networks.

These systems have a poor generalization capacity due to the use of heuristics. They imply many handcrafted features, requiring a priori knowledge from experts. It is costly to generate such systems and they hardly can adapt to data variability, from one language to another for example. Nowadays, the majority of HTR approaches still lies on a prior segmentation step, at word or line level, implying the recognition of whole words or lines.

3.1 Problem statement

Recognizing the text of a whole document is a difficult task, it involves many challenges. Not only does the system need to identify where the text is located in the image but it also needs to recognize it. In addition, the different identified text regions must be ordered in order to preserve the meaning of the text. This process is represented in Figure 3.1.

The most common way to tackle this task is to perform these three sub-tasks as three independent steps.

The segmentation stage

The text line segmentation task consists in localizing all the text lines from a given input image of a document. It involves many challenges. One does not know in advance the number of text lines to be recognized in the document. On top of that, they can start and

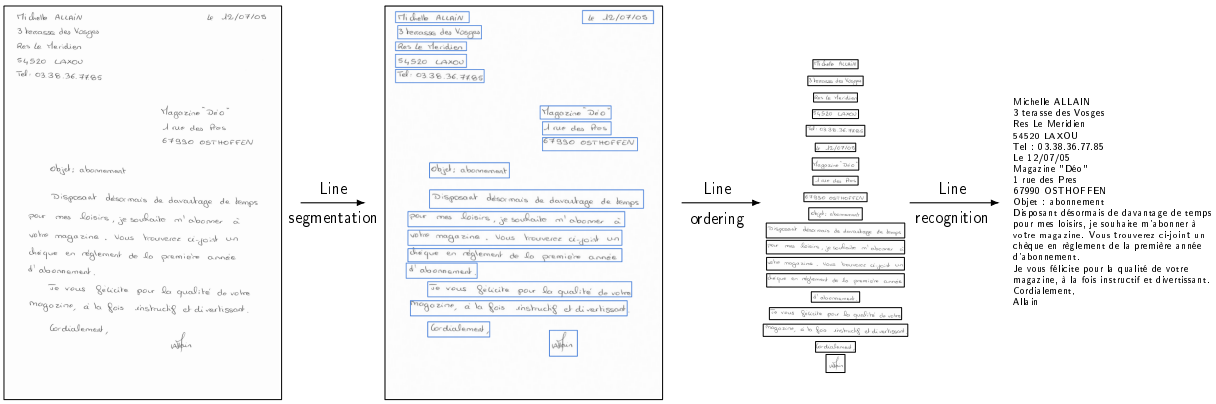
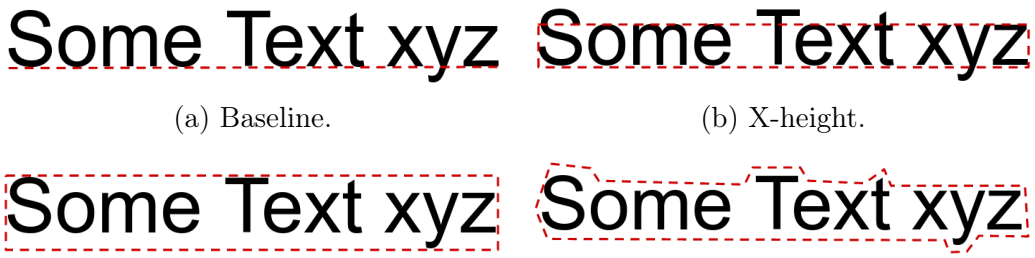


Figure 3.1: Three-step line-level approach for Handwritten Text Recognition.

end wherever in the document. As shown in Figure 3.1, the size of the text lines can vary a lot.

Using an explicit segmentation stage also raises the question of the definition of a line. Baseline, X-height, bounding box and polygon are examples of target labels for segmentation that have been frequently used in the literature, all with their pros and cons [90]. These different kinds of annotation are represented in Figure 3.2. In addition, due to the visual nature of this task, segmentation labels are prone to the subjectivity of the annotators, leading to heterogeneity in the labeling process.



(a) Baseline. (b) X-height.

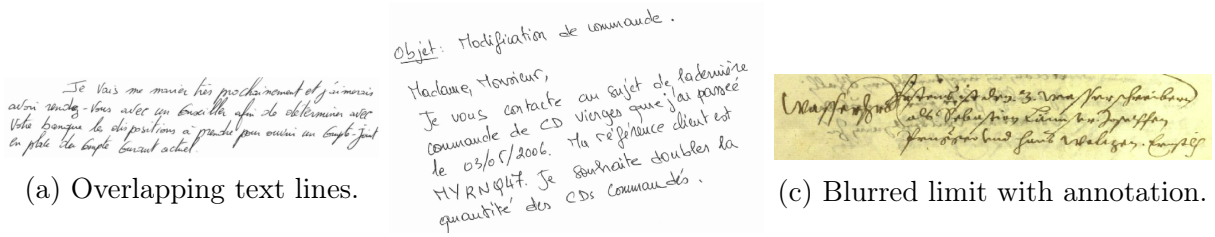
(c) Bounding box. (d) Polygon.

Figure 3.2: Examples of line segmentation annotations.

There are several cases in which the line segmentation can be difficult, as shown in Figure 3.3. Ascenders and descenders can lead to overlapping text lines; it results in noise coming from strokes of adjacent text lines. Text lines can be more or less slanted, preventing the use of strictly horizontal bounding boxes for instance. The last example shows how it can be complicated to distinguish the end of a text line and the beginning of another one in the context of multiple columns of text.

The ordering stage

The ordering step can be formalized as follows: the input is a set of text lines, whatever the line definition chosen, and the output is an ordered sequence of text lines. One can note



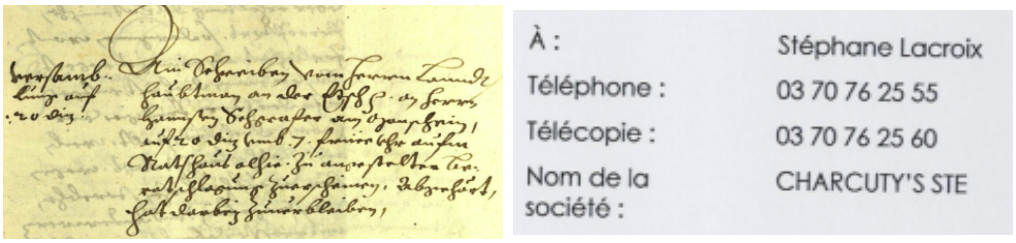
(a) Overlapping text lines.

(b) Slanted text lines.

(c) Blurred limit with annotation.

Figure 3.3: Line segmentation issues.

two main challenges for this task. First, there is not a single possible reading order for a given document. Indeed, for documents with a complex layout, such as tables or drawings, there are multiple ways of reading the textual content, while following a humanly logical order. The second challenge is about the need for the global context. Figure 3.4 illustrates this issue. For both images, there is only one correct reading order to preserve the global meaning of these documents. They both follow a similar double-column layout. However, the document on the left must be read column by column and the document on the right must be read row by row. Using only the segmented text line coordinates, it seems impossible to choose between those two orders. For such complex cases, it is required to have a global understanding of the document layout to understand the notion of margin or fields of forms. The textual content itself can also be helpful to choose how to order the text lines.



(a) Expected reading order by column.

(b) Expected reading order by row.

Figure 3.4: The line reading ordering depends on the global analysis of the document.

The recognition stage

The recognition step consists in recognizing the sequence of characters from a text line image. The main difficulty of this recognition stage is about the writing style variability: each person has his own way of writing with specific character shapes, slant, alignment and character spacing. The background and the quality of the input images can also dramatically affect the performance of the model. This difficulty is intrinsic to the task, it is not related to the technique used: it is the same for humans.

In addition, the recognition of text lines involves two main technical issues:

- A one-dimensional sequence of characters is expected as output whereas the input is a two-dimensional image.

- One does not know in advance the number of characters to be recognized, and it cannot be deduced from the width of the image neither.

Although handwritten text line recognition is *a priori* limited to visual character recognition, a Language Model (LM) is often associated, as post-processing, to improve the performance. A language model is a probability distribution over sequences of characters or words which represents a given language. The idea is to detect and correct sequences of characters or words that have low probability *i. e.* which are statistically incorrect. It is generally handled by a statistical approach [91, 92]. It consists in N-gram character or word language models which estimate probabilities over sequences of N successive characters or words. In this thesis, we focus on raw architecture performances, we do not study the use of such external language model.

3.2 Related works

We now present the related works for each of these three sub-tasks.

3.2.1 Text line segmentation

Early text line segmentation techniques [93] can be classified into three categories as suggested in [94]. First, the projection-based methods, which consider the boundaries between lines as valleys of vertical projection profile [95]. Second, the grouping methods; it consists in grouping rows of connected components according to heuristic rules [96]. The last category is the smearing methods which use blurring filters combined with binarization or active contours for example [97].

Nowadays, it is generally handled by deep neural networks. One can note two main approaches: pixel-level classification and object detection.

Pixel-level classification, in the context of text line segmentation, consists in classifying each pixel of the input image among two classes, namely background and text. It is generally handled by a Fully Convolutional Network (FCN) as in [90, 98, 99, 100, 101], as shown in Figure 3.5.

Since the prediction is carried out at pixel level, the prediction may present some scattered anomalies, even if the overall prediction seems correct. Some post-processings can be used to deal with this issue, through the use of heuristics based on neighboring predictions. The adjacent pixels labeled as text are grouped into blocks of pixels: each block corresponds to a text line. However, this is where the major drawback of this approach lies. Indeed, all the text line predictions are predicted in the same 2D representation. In the case of closely spaced text lines, the pixel-level predictions of successive text lines could touch and merge: the segmentation fails in such cases.

The object-detection approach, through the prediction of bounding box coordinates, solve this issue. This approach is depicted in Figure 3.6. The models proposed in [102, 103, 104] follow this approach at word level. A CNN is used to extract features from the input image. It is followed by a Region Proposal Network (RPN). The idea is to assign an anchor point to certain coordinates in the image, following a sliding window approach, to cover the whole

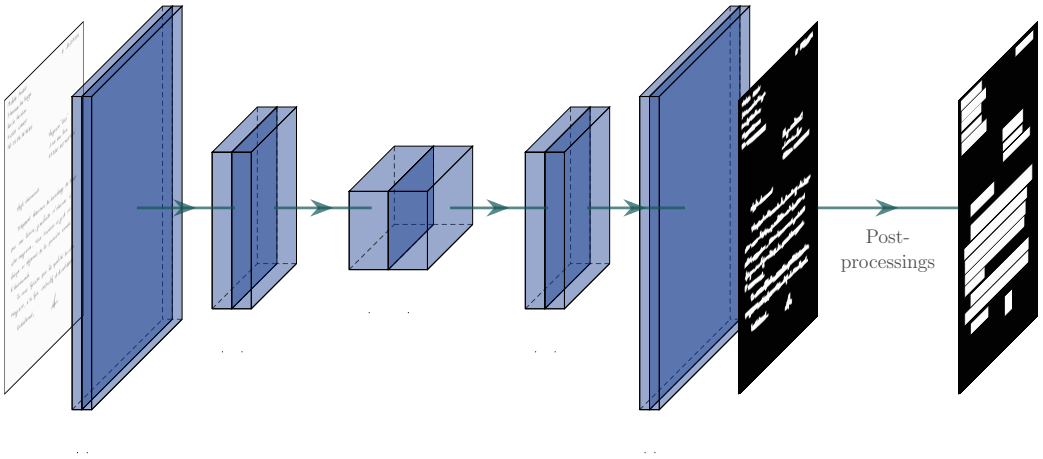


Figure 3.5: Line-level segmentation by pixel-level classification (FCN).

image. It leads to a grid of anchor points. These anchor points correspond to the center of potential regions containing a word. For each anchor, rectangles of varied shapes and sizes are predefined, leading to a high number of potential bounding boxes. For each one, offsets are predicted to give more flexibility in the prediction of the bounding box coordinates. This region proposal network is combined with a non-maximal suppression process in order to filter these proposals. It is an iterative process in which proposals with highest confidence scores are kept and overlapping one are discarded. The remaining proposals are the final predicted word bounding boxes.

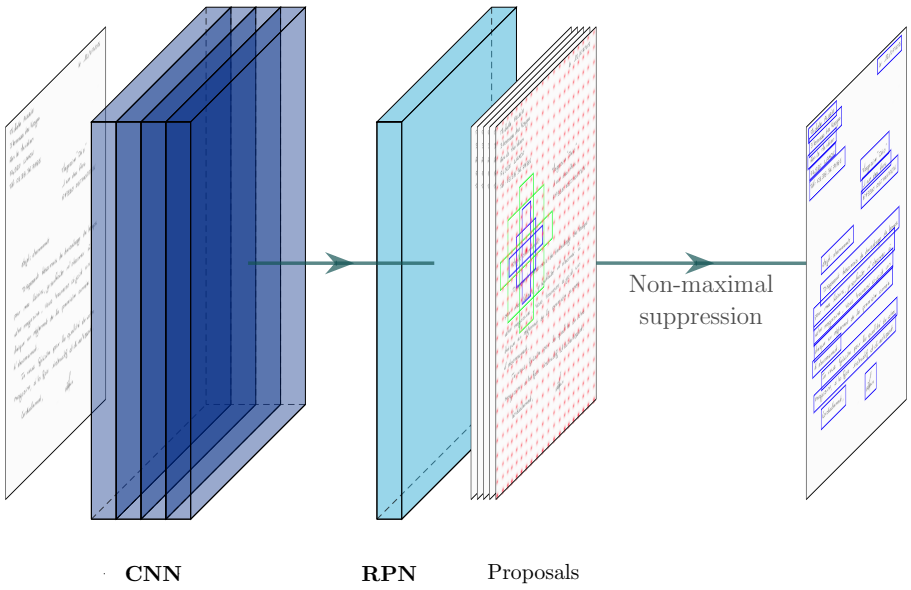


Figure 3.6: Line-level segmentation with an object-detection approach (RPN). Anchors are represented by red dots and an example of proposals for a single anchor is depicted by green and blue rectangles.

In [102, 103], the authors propose such a model following a multi-task end-to-end architecture. The segmentation part is jointly trained with the word recognition part. In [104], the word bounding boxes are merged into line bounding boxes using some rules based on heuristics: the recognition is carried out at line level. In [105], the authors proposed to separately detect bottom-left and top-right corners of text line bounding boxes, using an additional matching step.

The idea of region proposal is also used in [106, 107, 108]. However, these approaches do not attempt to predict text line bounding boxes but only the coordinates of the start position of the text lines, as well as their height. In [106], a CNN+MDLSTM is used to predict these start-of-line references. The line width is considered as the image width. A specific end-of-line token is added for the recognition part to handle multi-column texts. In [107, 108], a CNN is used as start-of-line predictor. Then, a recurrent process predicts the next position based on the current one until the end of the line, generating a normalized line. The idea is to deal with curved lines. The approach proposed in [108] is similar to the one of [107], but it can handle transcriptions without line breaks.

3.2.2 Text line ordering

Very few works proposed whole line-level HTR pipeline: text line segmentation and text line recognition are mostly processed independently. This way, the text line ordering aspect has not been much studied or detailed in the literature. It seems that the majority of the works focused on a rule-based ordering approach. The reading order is computed based on the coordinates of the text regions and some heuristics. Generally, for Latin languages, the reading order is fixed from top to bottom and from left to right. It is sufficient for datasets in which the layout is regular. It becomes tedious for heterogeneous documents. Very few works have been working on learning-based approaches for text line ordering: [109, 110]. In these works, the authors considered the problem as a binary relation ordering issue between each pair of text lines.

3.2.3 Text line recognition

Regarding handwritten text line recognition, it was first solved using handcrafted features and HMM [111, 112]. However, those models lacked discriminative power and were limited when dealing with long term dependencies in sequences. For a long time, hybrid systems combining HMM with neural networks were proposed, leading to better results over standard HMM, thanks to the discriminative power of neural networks: HMM+MLP [113, 114, 115, 116], HMM+CNN [117] or HMM+RNN [118, 119]. Currently, the state of the art is reached with deep neural networks, without the use of HMM anymore.

The recognition of whole words or lines by deep neural networks was mainly made possible by the Connectionist Temporal Classification (CTC) to handle the alignment issue between the one-dimensional predicted sequence and the one-dimensional ground truth sequence, which do not have the same length (issue previously handled by HMM). However, it remains a main challenge: the input is a two-dimensional image whereas the expected output is a one-dimensional sequence of characters. This way, HTR architectures are usually made up of three components: an encoder, a transformation mechanism and a decoder. The encoder

aims at extracting features from the input image. The transformation mechanism is used to go from a two-dimensional feature space to a one-dimensional feature space. The decoder enables to associate to each feature frame a probability for each character of a given alphabet, as well as a probability for the CTC null symbol (blank).

In 2008, A. Graves *et al.* proposed to use a Multi-Dimensional Long-Short Term Memory (MDLSTM) as encoder for text line recognition [120] so as to model the dependencies of the whole input image. The transformation mechanism consists in summing all rows of the features maps element-wise, to collapse the vertical dimension, leading to a one-dimensional sequence. The decoder is only a softmax activation. With the emergence of Convolutional Neural Network (CNN) and their efficiency for computer vision tasks, the authors of [121] proposed to alternate between convolutional layers and MDLSTM layers (CNN+MDLSTM). The same idea is studied in [122]. An example of CNN+MDLSTM architecture is depicted in Figure 3.7.

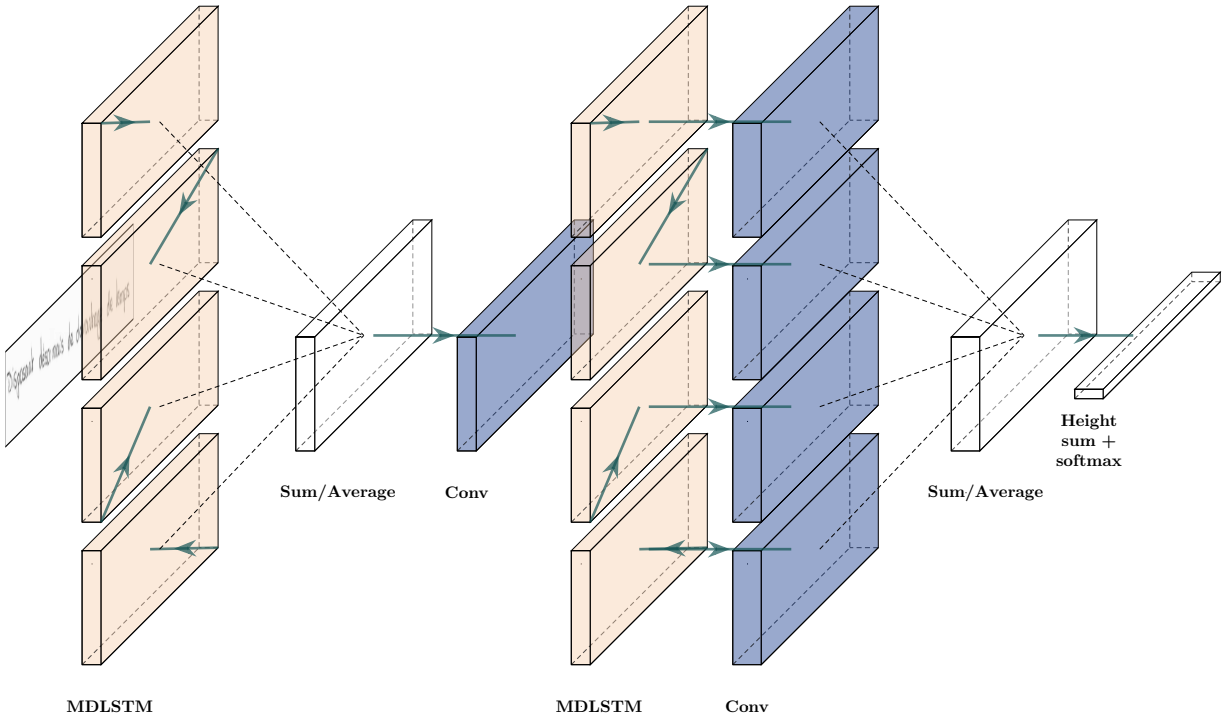


Figure 3.7: CNN+MDLSTM architecture for text line recognition.

One should note that recurrent layers present a main drawback: the computations are sequential, leading to longer training and prediction times. On the other hand, the computations of the convolutional layers are parallelizable and the algorithms have been highly optimized to this end. In addition, for the specific case of Long-Short Term Memory (LSTM) cells, it can lead to an important number of trainable parameters.

It leads to a trend toward the reduction of the use of recurrent layers. In 2017, the authors of [123, 124] proposed to use a CNN as encoder and a BLSTM as decoder (CNN+BLSTM), as shown in Figure 3.8. In [124], the transformation mechanism is a flatten operation over the vertical dimension. The authors of [125] argued that MDLSTM would provide better

results for complex handwriting compared to CNN+BLSTM while the performance would be equivalent for simpler ones.

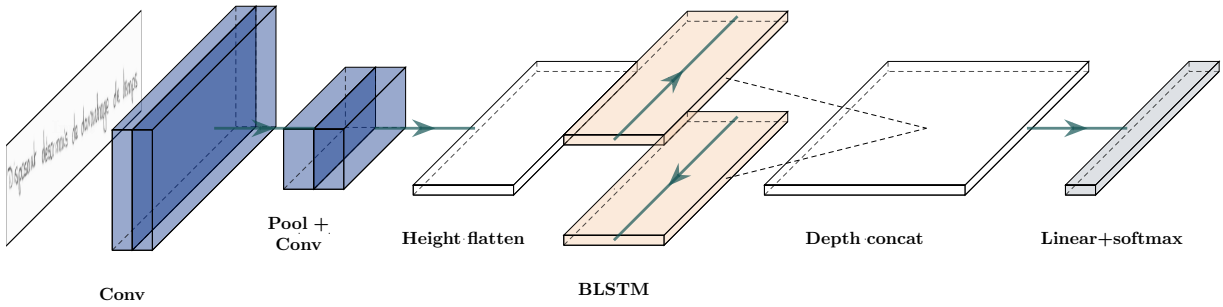


Figure 3.8: CNN+BLSTM architecture for text line recognition.

Based on the successful application of gates in [63] for language modeling and in [64] for Neural Machine Translation (NMT) in 2017, the authors of [126] and [127] proposed Gated Convolutional Recurrent Network (GCRL): CNN+BLSTM architectures with gates in the convolutional part. The idea is to counteract the decrease in the number of LSTM layers, which integrate gates to select information along the time axis, by integrating this information flow control over the depth axis. Following this idea, we proposed a Gated Convolutional Neural Network (GCNN) in 2019, removing every recurrent layers. The same year, [128] achieved state-of-the-art results on isolated words on the IAM dataset, using a CNN without any gate component. Figure 3.9 shows an example of CNN architecture for line-level HTR.

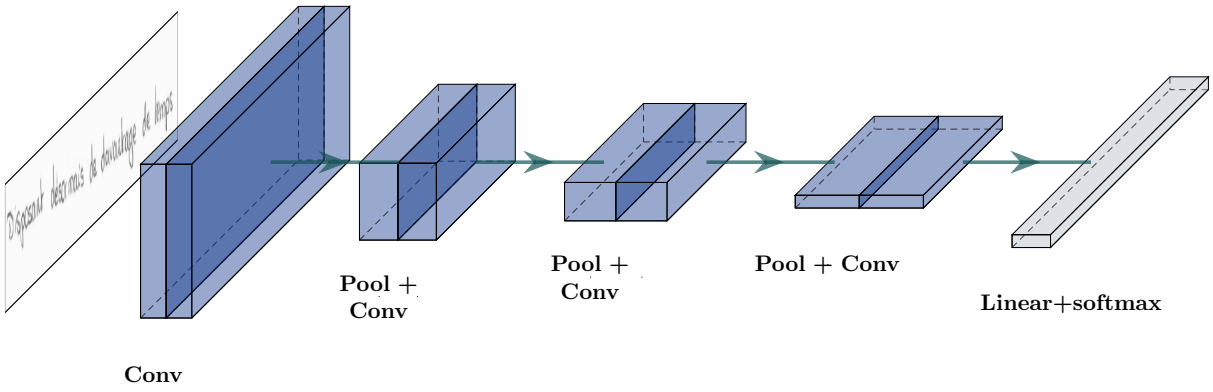


Figure 3.9: CNN architecture for text line recognition.

The authors of [129] proposed a Gated Fully Convolutional Network (GFCN) *i. e.* a GCNN without any dense layer, reaching state-of-the-art results on many datasets, including IAM at line level. Their architecture consists in a large stack of convolutional layers, leading to 26 million of trainable parameters.

All the works we have just mentioned use the CTC loss to train their models. However, based on [58] and the emergence of attention models, [130] and [131] successfully applied

attention-based CNN+BLSTM to the task of HTR at line level. The models follow a recurrent process, generating the output character by character. They use specific <start-of-line> and <end-of-line> tokens to initialize and stop the recurrent process, respectively. The main drawback of this approach lies in the sequentiality of the process, inherent to the recurrence, which leads to prediction time dependent on the sequence length. The authors of [132] also proposed an attention-based approach. However, while in [130] and [131] the features are flattened before the attention mechanism, in [132], the features are still in two dimension and the attention weights are computed over the horizontal axis only: the same weight is applied for each vertical position of a same column.

Inspired by these previous works, we proposed a GFCN, in 2020, to take advantage of the parallelization of the computations, with the aim of having fast training and prediction times, with few trainable parameters. As a matter of fact, despite the use of convolutional components only in [129], the high number of parameters reduces the advantage of using them by increasing the computation time. We now present the GFCN model we proposed.

3.3 Gated Fully Convolutional Network for Handwritten Text Line Recognition

We propose a Gated Fully Convolutional Network for the task of HTR at line level. Trained with the CTC loss, the model reached competitive results on the RIMES and IAM datasets at line level.

In this section, we introduce the architecture of the proposed GFCN and we provide an experimental study to evaluate the performance of this model in the context of text line recognition. We provide all source code and pre-trained model weights at `https://github.com/FactoDeepLearning/LinePytorchOCR`.

3.3.1 Architecture

The aim of the proposed network is to combine the following expected properties: a large receptive field, to efficiently model the dependencies in the input space, a small number of parameters, to generalize well and for computational efficiency, and the only use of convolutional components for computation parallelization. These properties are obtained by a deep FCN architecture enhanced by a strong gating mechanism and by the use of Depthwise Separable Convolution (DSC).

The proposed architecture is a GFCN inspired from our preliminary work on GCNN applied to handwritten text recognition [133]. In other words, it is only made up of convolutional components: convolutional and pooling layers, enabling it to deal with input images of various sizes. No recurrent nor fully connected layers are used.

We opted for a GFCN because convolutional layers are light components: operations can be parallelized on GPU and they do not require a lot of parameters. Therefore, we can stack many of these layers without major impact on memory usage or training and prediction times. The proposed model contains 22 convolutional layers, enabling to reach a receptive field of size (v=196, h=240) where v and h stand for the vertical and horizontal dimensions,

respectively. We mostly use DSC to reduce the number of parameters. As defined in [41] (Section 2.2.2.1), DSC consists in performing a depthwise spatial convolution followed by a pointwise convolution. The advantage is that these operations need less trainable parameters than standard convolutions, while providing comparable results. However, we keep standard convolutional layers at the beginning and at the end of our model because these layers are more crucial to extract features and predict probabilities. Through our experiments DSC turned out to be less efficient when introduced on these layers.

We used another component which is the Max Pooling. It enables to increase the size of the receptive fields while reducing the tensors size and thus the memory usage, preserving the most relevant information. We used it until reducing the vertical dimension to unity while only dividing by four the horizontal dimension so as to keep enough frames to align the horizontal representation with the ground truth using the CTC loss.

The full model is depicted on Figure 3.10. In order to provide a better view of the proposed model, we have gathered some layers into GateBlocks and ConvBlocks. The GFCN takes as input images of 64 pixels in height with variable width. The model starts with 2 ConvBlocks of respectively 32 and 64 filters. It preserves the original image size to extract low level details. It is followed by 5 GateBlocks, which progressively decrease the height and the width. A DSC with 2×1 kernels is then used to collapse the vertical dimension, leading to a one-dimensional sequence of feature frames. A succession of 6 DSC+Gate+Dropout is applied with 1×8 kernels to enlarge the receptive field along the horizontal axis. A last convolution with 1×1 kernels, combined with a softmax activation, enables to provide the probabilities of each character (N being the charset size + 1 for the CTC blank) for each output frame. The model is trained in an end-to-end fashion with the CTC loss.

ConvBlocks

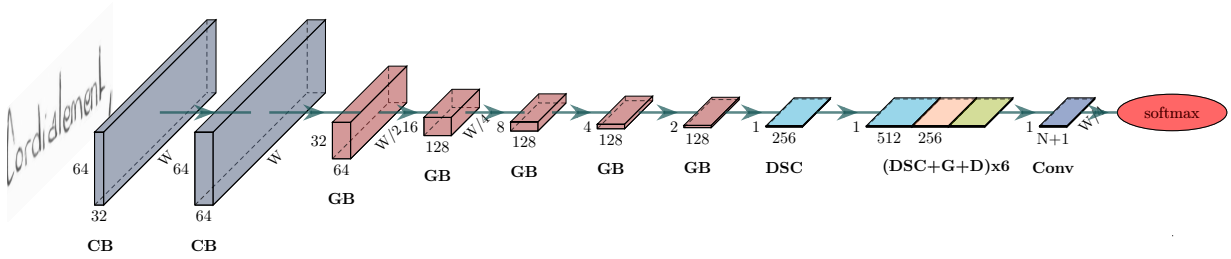
A ConvBlock consists in 2 standard convolutions with C filters, followed by instance normalization and dropout with a probability of 0.4. Each convolution has the following parameters: 3×3 kernel, 1×1 stride, 1×1 padding and ReLU activation. A visualization of a ConvBlock is provided on Figure 3.10b.

GateBlocks

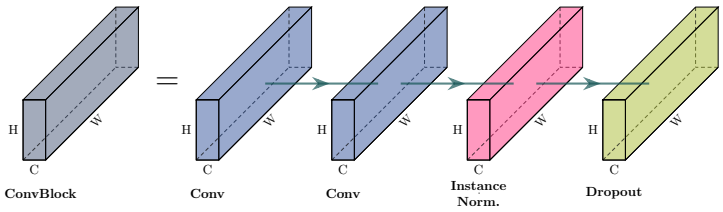
A GateBlock is defined as a succession of two DSC (C filters, 3×3 kernel, 1×1 stride, 1×1 padding, each followed by a ReLU activation) followed by instance normalization, Max Pooling, Gate and dropout layers. For the two first GateBlocks, Max Pooling is carried out with a kernel size of 2×2, the three following blocks have a kernel size of 2×1. It means that the height is divided by 32 and the width by only 4. Dropout is applied with a probability of 0.4. Figure 3.10c shows a GateBlock in a more visual way.

The gating mechanism

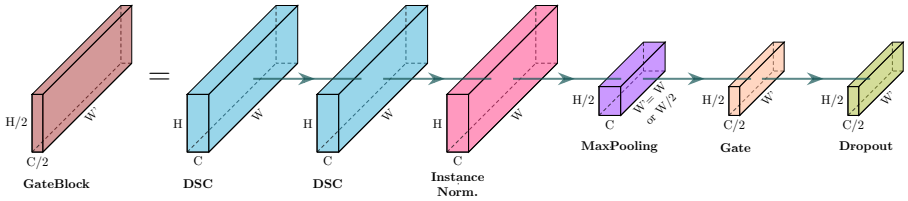
The model integrates the concept of Gate which enables to select the relevant features throughout the layers. The gating mechanism is inspired from [129] and is defined as follows: the input tensor is split over the channel axis into two sub-tensors of same dimensions. A tanh activation is applied to one sub-tensor and a sigmoid to the other. Both are then layer



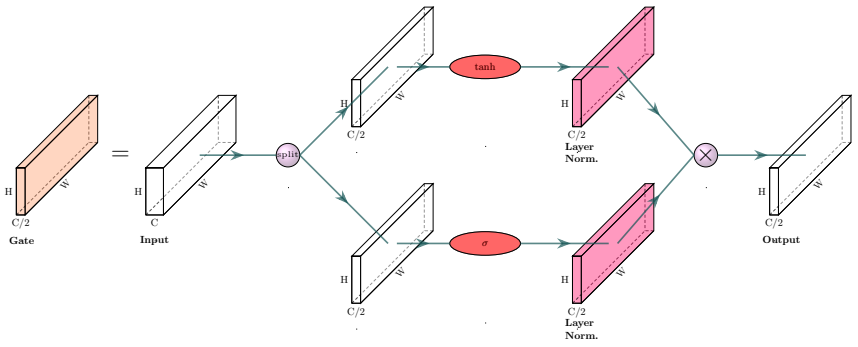
(a) Overview of the GFCN.
CB: ConvBlock, GB: GateBlock, G: Gate, D: Dropout.



(b) ConvBlock (CB).



(c) GateBlock (GB).



(d) Gating mechanism (G). White elements are just representations of the tensors at a given step (no specific operation is performed).

Figure 3.10: Proposed GFCN architecture. The model is made up of some ConvBlocks and GateBlocks, and relies on a gating mechanism.

normalized separately before being multiplied element-wise. Thanks to the sigmoid function, the gate can be seen as a selection operator which is used over the layers. A visualization of this mechanism is given on Figure 3.10d.

Regularization techniques

Regularization techniques are used to improve the training stability and to reduce overfitting. In this way, the architecture starts with a gaussian noise layer. Instance normalization is added after the convolutional layers in the ConvBlocks and the GateBlocks, and dropout is also applied multiple times. We choose to use the instance normalization instead of batch normalization which is widely used in the literature because it is independent of the mini-batch size and it showed great results in other tasks such as stylized image generation [34]. As a matter of fact, since we used small mini-batches, the batch normalization is not that stable in our case. This choice is analyzed in Section 3.3.2.

3.3.2 Experimental study

The experimental environment is as follows:

- Datasets: to evaluate the proposed model, we used two public datasets at line level: RIMES 2011 (line-2) and IAM, as defined in Section 2.5.1.
- Pre-processings: images are resized to obtain a height of 64 pixels conserving their initial width. Mean and variance are computed over the training data set to normalize the grey-level pixel intensity.
- Training details: the experiments are carried out with Pytorch, using a single GPU V100 (16 Go). We used the CTC loss to train the model. We used Adam optimizer with an initial learning rate of 10^{-4}. Mini-batches of size 2 are used during training and evaluation. Trainings are stopped if no improvement is observed during 50 consecutive epochs. We do not use any data augmentation technique in the experiments. We do not use any external data nor external LM.
- Metrics: we used Character Error Rate (CER) and Word Error Rate (WER) as metrics to evaluate the performance of the proposed approach, as detailed in Section 2.5.2. For the WER, punctuation marks are not counted as words: they are considered as belonging to the previous word.

We evaluate the proposed GFCN on both RIMES and IAM datasets and compare it to the state-of-the-art approaches. Then, we discuss the normalization technique used and the impact of the receptive field through experiments.

Comparison with state-of-the art handwritten text line recognition systems

The following comparisons are made with approaches under similar conditions, *i. e.* at line level, without the use of external data, data augmentation strategy, lexicon constraints nor external LM.

Table 3.1 shows the state-of-the-art results on the RIMES datasets. As we can see, the proposed model reaches better results than some recurrent architectures even if it is not at the leading position. The proposed GFCN reaches a CER of 4.35% on the test set, which is not far from the Puigcerver CNN + LSTM model that reaches the best CER with 3,3%. One can notice that, any of the architectures (CNN+BLSTM, CNN+MDLSTM and GFCN) reach similar CER.

Table 3.1: Comparison of the GFCN results with the state of the art on the RIMES dataset without LM, lexicon, nor data augmentation.

Architecture	CER (%) validation	WER (%) validation	CER (%) test	WER (%) test
CNN+MDLSTM [125]	3.32	13.24	4.94	16.03
CNN+MDLSTM-X2 [125]	3.14	12.48	4.80	16.42
CNN+BLSTM [125]	**2.9**	11.68	4.39	14.05
CNN+BLSTM [124]	3.0		**3.3**	
Ours (GFCN) [134]	3.82	15.60	4.35	18.01

The results obtained on the IAM dataset are presented in Table 3.2.

The proposed model reaches 7.99% of CER which is very close to the best recurrent model that achieves 7.73%.

Table 3.2: Comparison of the GFCN results with the state of the art on the IAM dataset without LM, lexicon nor data augmentation.

Architecture	CER (%) validation	WER (%) validation	CER (%) test	WER (%) test
CNN+MDLSTM [125]	5.41	20.15	8.88	29.15
CNN+MDLSTM-X2 [125]	5.40	20.40	8.86	29.31
CNN+BLSTM [124]	5.1		8.2	
CNN+BLSTM [125]	**4.62**	17.31	**7.73**	25.22
Ours (GFCN) [134]	5.23	21.12	7.99	28.61

Comparison of the models

We now compare the previously mentioned models by considering additional features, and not only the performance (CER): we also consider the training time, the prediction time and the number of parameters at stake. We have reproduced the best model seen previously to get those results, for the others (CNN+MDLSTM and CNN+MDLSTM-X2) we only give the number of parameters. This experiment was conducted on the IAM dataset with images resized to 128 px height and preserving the original width. We added one GateBlock (and thus one Max Pooling layer) to our model in order to be compatible with images of that height. The results are presented in Table 3.3. Training time corresponds to the time spent to train over a full epoch (train set) and prediction time is the mean time to predict a sample of the IAM test set. Both are computed with a mini-batch size of 2 on a GPU V100 32Go.

As we can see among the best models, ours is the one with the lowest number of parameters. Training time and prediction time of the CNN+BLSTM [124, 125] have the same

Table 3.3: Comparison of the GFCN with the state-of-the-art architectures for the IAM dataset, for an input image height of 128px, preserving the original width.

Architecture	Training time (min/epoch)	Prediction time (ms/sample)	Parameters
CNN+MDLSTM [125]			0.8 M
CNN+MDLSTM-X2 [125]			3.3 M
CNN + CNN+BLSTM [124, 125]	11.25	57	9.6 M
Ours (GFCN) [134]	13.75	74	1.4 M

order of magnitude than that of the proposed GFCN. This can be explained by the high number of normalization layers used in our model and its depth that counterbalance with the sequential computations of the LSTM layers of the other models. It has to be noted that CNN+MDLSTM and CNN+MDLSTM-X2 architectures performed well too and they do not imply so many parameters. That is especially true for the CNN+MDLSTM which uses the smallest number of parameters of all of the reported models.

Comparison of the normalization layers

In this experiment, we are looking for the best normalization technique for the proposed model. In this respect, we trained the model on the RIMES dataset with different normalization layers in place of the instance normalization layers shown in Figure 3.10. We tested the 4 most common normalization techniques namely batch normalization [32], layer normalization [35], instance normalization [34] and group normalization [36] for a group of size 32. Results are shown in Table 3.4. Reported CER are the best CER on the validation set over the first 50, 100, 150 or 200 epochs. The training time for one epoch is also given since it can be an important criterion.

Table 3.4: Comparison of the different kinds of normalization for the proposed GFCN with the RIMES dataset. CER is computed on the valid set.

Normalization	CER (%) 50 epochs	CER (%) 100 epochs	CER (%) 150 epochs	CER (%) 200 epochs	Time (/epoch)
Instance	6.87	**5.03**	**4.47**	**4.28**	**8.5 min**
Layer	**6.75**	5.04	**4.47**	**4.28**	15 min
Group (32)	7.10	5.30	4.86	4.32	8.75 min
Batch	9.6	5.7	5.4	4.8	**8.5 min**

As one can see, from 100 epochs, the best CER is obtained with instance normalization. Layer normalization is acting similarly to instance normalization but it implies longer training time which makes it less relevant. Group normalization performs well too with tiny differences. Batch normalization is the one with the worst CER. This can be explained by the small mini-batches used that lead to a slightly less stable batch normalization. As a matter of fact, it is the only normalization that is mini-batch size dependent. The others use the samples in an independent way to compute means and variances. Given these results we can conclude that, apart from batch normalization, any of these normalization techniques can be considered. Batch normalization should be preferred with larger mini-batches only.

Instance normalization turned out to be an option to be considered when dealing with images and small mini-batches; that's why we kept this one.

Impact of the receptive field

This last experiment aims at determining the impact of the receptive field in a GFCN architecture. In this respect, we vary the number of (DSC + Gate + Dropout) used in our model which is fixed to 6 in our baseline. We made this number vary from 1 to 6; for each one we kept the best CER on the valid set over the first 100 and 200 epochs and we report the respective number of parameters and receptive field. The results are presented in Table 3.5.

Table 3.5: Impact of the receptive field on the GFCN results, on the IAM dataset. CER is computed over the valid set.

Number of ending Gates (DSC+G+D)	CER (%) 100 epochs	CER (%) 200 epochs	Parameters	Receptive Field (h, w)
6 (baseline)	6.82	**5.80**	1,375,792	(196, 240)
5	**6.69**	5.97	1,241,904	(196, 212)
4	8.14	7.48	1,108,016	(196, 184)
3	6.93	6.23	974,128	(196, 156)
2	7.35	6.63	840,240	(196, 128)
1	8.30	7.83	706,352	(196, 100)

One can notice that only the horizontal axis is affected by this number since the DSC has a kernel 1×8. Moreover, we double the horizontal receptive field from 1 to 6 (DSC+G+D) which switch from 100 to 240. In the same way, the number of parameters is almost doubled (from 0.7 to 1.4 million).

Considering that characters have an average width of 31 pixels on the IAM test set, the network can roughly model dependencies through 3 successive characters for prediction when using only one (DSC+G+D). For the baseline, the receptive field corresponds to 8 characters. We may assume that this very large receptive field enables to compensate the memory capability of LSTM.

We can easily see a tendency whereby the CER is improved when increasing the number of (DSC+G+D) and thus the receptive field. We may assume that the high CER of example 4 is due to a bad initialization that would be corrected with more epochs. It would be interesting to cross validate these results by repeating this experiment multiple times to consolidate this assumption. Here, doubling the receptive field this way enables to reduce the CER by 2 points at 200 epochs going from 7.83% down to 5.80%.

3.3.3 Discussion

We have presented one of the first Gated Fully Convolutional Networks applied to handwriting line recognition. The proposed model reaches competitive results on the RIMES and IAM datasets, compared to the best recurrent models in the same conditions. This demonstrates that recurrence-free models should be considered for the task of text recognition. The use of DSC enables to have very few trainable parameters at stake, and the selective

mechanism provided by the gates, combined with the regularization techniques enabled to get a stable deep neural network. One can notice that the vertical receptive field is larger than the height of the text line images (196 compared to 64). The idea behind this GFCN network is to have a generic encoder that could then be used to process whole paragraph or page images, as we will see in the next chapters.

However, although we only used convolutional components, which enables highly parallelizable computations, the high stacking of them, combined with the normalization layers, leads to higher prediction time when compared with the standard CNN+BLSTM architecture.

With some hindsight, the effectiveness of the gates has yet to be proven. Indeed, we propose a new FCN architecture in the next chapter for which the use of this gating mechanism was not beneficial anymore. It has to be noted that this latter architecture was trained with data augmentation, contrary to the GFCN. In addition, the FCN relies on residual connections, which may compensate for the contribution of the gates. For this reason, we do not use any gating mechanism in the next architectures.

3.4 Conclusion

Since this contribution, one can notice a recent trend towards the use of attention-based architectures for HTR at line level, and more specifically based on transformers [65]. In [135] and [136], the authors proposed some architectures made up of a CNN + transformer for encoder and transformer for decoder. They are trained with the cross entropy loss using specific <start-of-line> and <end-of-line> tokens. The authors of [136] showed that the addition of a LM is useless for their model *i. e.* the transformer architecture enabled to model dependencies between the predicted characters. In [135], the authors proposed a bidirectional approach for the transformer architecture, combined with a voting algorithm, to reduce the error rate. The authors of [137] conducted an extensive study to compare several architectures for text line recognition, including CNN+transformer encoder and GCRL, both with CTC decoding or transformer decoding. They found out that, in their configuration, *i. e.* when using an external dataset and an external LM, the combination of CNN+transformer encoder trained with the CTC loss is better.

The proposed models are more and more efficient. In [137], their best model reached 2.75% of CER on IAM and 1.99% of CER on RIMES at line level, when combined with external data and language models. This is extremely encouraging even if there is still room for improvement. However, one should keep in mind that all these results are given for images of isolated text lines which have been manually annotated or at least manually verified. In real world, handwritten documents are nearly never limited to a single text line: the metrics are biased and does not meet the main use case.

Although in the majority in the literature, the three-step approach (segmentation then ordering then recognize) has three main drawbacks:

- The errors accumulate *i. e.* if a text region is poorly segmented, it will inevitably lead to recognition errors. And even with a perfect segmentation stage, errors can occur during the recognition step. This way, metrics should take into account the prior segmentation and ordering steps which themselves may include some errors.

- It requires additional annotations. Since the aim is to recognize the textual content of a document, the minimum requirement is the transcription annotation. Reading order can be deduced from this transcription. However, the segmentation stage requires its own additional annotations, which are difficult to define (see Section 3.1) and costly to produce.
- Each stage of the three-step approach being independent, they cannot benefit each other to improve their results. Yet, it is well known that recognition and segmentation should be performed altogether so as to explore and evaluate all the hypotheses as a whole [138]. Obviously, the same goes for reading order, since recognition could help a lot to search for the best hypothesis.

One way to alleviate these issues is to recognize whole handwritten paragraphs, thus reducing the number of segmentation entities per document required to train the system. We study these paragraph-level approaches in the next chapter.

Chapter 4

Handwritten paragraph recognition

Historically, the methods used for the Handwritten Text Recognition (HTR) task have gradually improved, reducing the need for the segmentation step. We have moved from character-level segmentation to word-level and then line-level segmentation, leading to better and better results. The intuitive next step is the paragraph-level segmentation coupled with an end-to-end paragraph recognition approach.

4.1 Problem statement

First of all, let's define what we mean by paragraph. In this thesis, paragraph refers to a subpart of the input document image made up of a set of text lines organized as a single column of text. Paragraph-level recognition follows the same steps as line-level text recognition: segmentation, ordering and recognition, as depicted in Figure 4.1. It means that paragraph-level recognition keeps the drawback of the errors which accumulate between these various stages. However, the main advantage of processing the input document at paragraph level is the reduced need for segmentation annotations. For the example presented in Figure 4.1, the number of bounding boxes for the segmentation stage decreased from 19 for the line-level approach down to 8 for the paragraph-level approach. In addition, the model can benefit from a larger context and one can expect to get improved predictions.

However, when switching from line images to paragraph images, one faces new challenges:

- The number of lines varies from one paragraph to another, knowing that each line contains an unknown and variable number of characters.
- The layout diversity can be important due to multiple factors such as interline spacing, horizontal alignment or slant.
- The recognition of whole paragraphs adds a new level of complexity regarding the reading order. While line-level recognition approaches only rely on a horizontal reading order for Latin languages, paragraph-level recognition approaches also rely on a vertical reading order: from one line to the next.

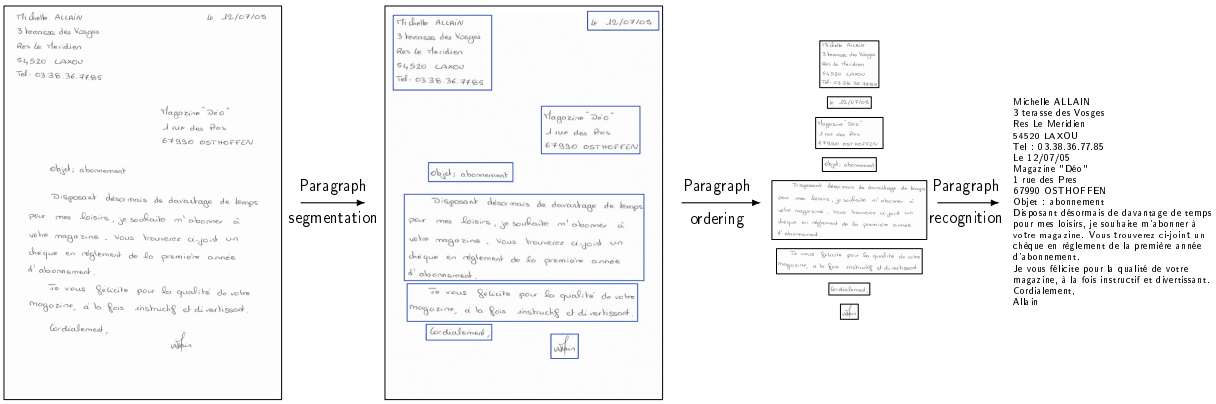


Figure 4.1: Paragraph-level approach for Handwritten Text Recognition.

4.2 Related works

In the literature, only very few works have been devoted to paragraph-level recognition, and most studies have concentrated on line-level recognition, as detailed in Chapter 3. In addition, to our knowledge, there is no work proposing the full three-step pipeline: the literature only focus on the segmentation stage or on the recognition stage, separately.

4.2.1 Paragraph segmentation

The paragraph segmentation stage has not been studied as single goal. It is mostly part of a more generic task, named Document Layout Analysis (DLA). DLA aims at identifying and categorizing the regions of interest in a document image. In [139], the authors proposed a method based on connected components to recognize 8 types of objects on heterogeneous documents: text, photographic image, hand drawn line area, graph, table, edge line, separator and material damage. The authors of [140] proposed a 2-stage method based on an artificial neural network for the semantic segmentation task, applied on historical handwritten documents. It detects multiple zone types such as page numbers, marginal notes and main paragraphs.

Nowadays, as for line-level segmentation, DLA is mainly handled through pixel-by-pixel classification using Fully Convolutional Networks (FCNs) ([141, 98, 142]). It consists in an elegant and relatively light end-to-end model that does not require to re-scale the input images. The authors of [142] applied DLA on printed textual documents: contemporary magazines and academics papers. They trained their model to recognize multiple classes namely figures, tables, section headings, captions, lists and paragraphs. The model presented in [98] aims at detecting different items from historical documents: text regions, decorations, comments and background. In [141], the model is applied to historical newspapers. It recognizes many textual elements such as titles, text blocks and advertisements, as well as images.

4.2.2 Paragraph recognition

End-to-end approaches for paragraph recognition can be classified into two categories: the one-shot approaches, which predict the whole sequence of characters in a single step, and the attention-based approaches which are based on a recurrent process.

4.2.2.1 One-shot approaches

As for many line-level approaches, one-shot paragraph-level approaches use the CTC loss and decoding process to handle the length variability between the predictions and the ground truths. However, contrary to isolated text lines, paragraphs can contain multiple lines of text and one cannot collapse the vertical dimension columns by columns because it would lead to only one predicted character per column of pixels.

The authors of [143] proposed a two-dimensional version of the CTC to tackle this problem: the Multi-Dimensional Connectionist Classification (MDCC). Using a Conditional Random Field (CRF), ground truth transcription sequences are converted into a two-dimensional model (a 2D CRF) able to represent multiple lines, as shown in Figure 4.2. A line separator label (¶) is introduced in addition to the CTC null symbol (∅). A single white space label (␣) is added at the beginning and at the end of each text line. The CRF graph enables to jump from one line to the following one, whatever the position in the current line. As for the CTC, repeating labels only account for one. An MDLSTM-based network is used to generate probabilities in two dimensions, preserving the spatial nature of the input image. The decoding process is carried out line by line. An example of prediction and correct path (solid line) for the first text line is depicted in Figure 4.2.

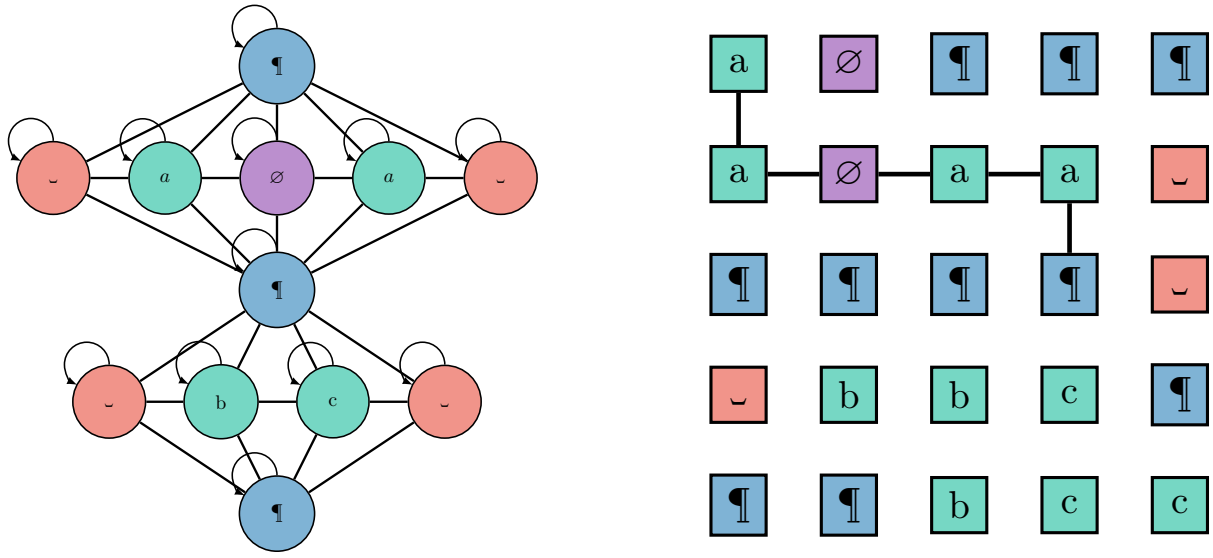


Figure 4.2: Multi-Dimensional Connectionist Classification. Left: ground truth representation for the text "aa\nbc". Right: prediction example.

Contrary to this approach, which aims at dealing with 2D predictions by proposing a new loss, in [144], the aim is to reformulate the two-dimensional problem as a one-dimensional

problem in order to use the standard CTC loss. Indeed, the authors of [144] focused on learning a representation transformation to unfold the input paragraph image into a single text line. The system is trained to concatenate text line representations to obtain a single large text line, before character recognition takes place. This is mainly carried out with bi-linear interpolation layers combined with an FCN encoder, as shown in Figure 4.3. This transformation network enables to use the standard CTC loss and to process the image in a single step.

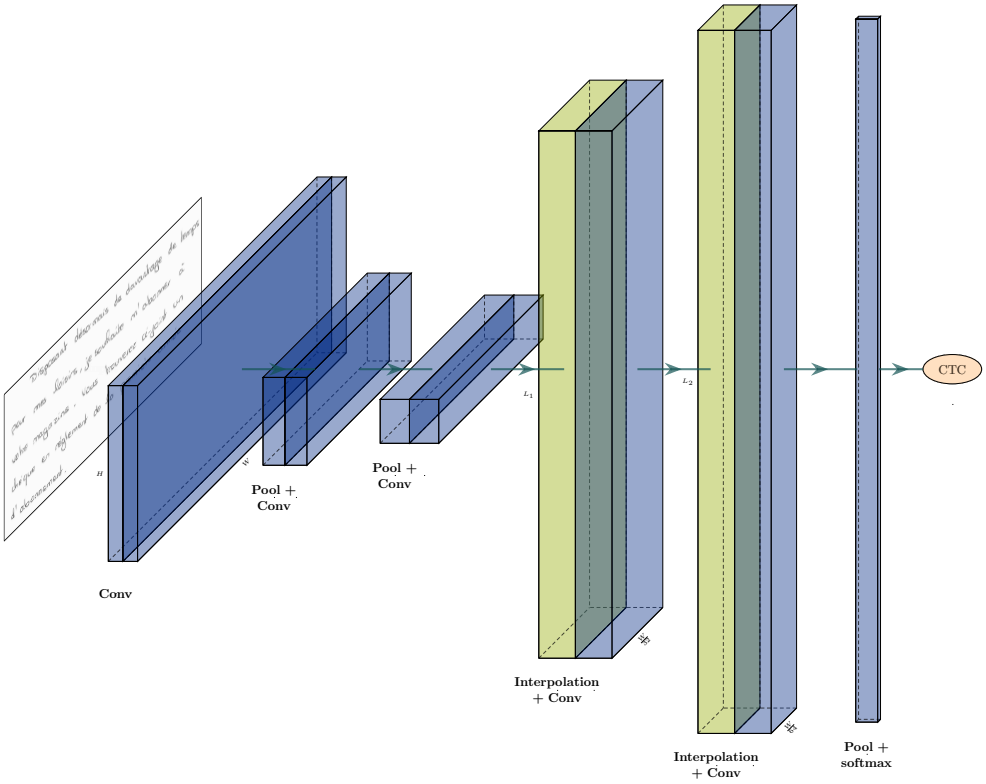


Figure 4.3: OrigamiNet overview.

4.2.2.2 Attention-based approaches

Attention-based approaches have also been proposed for end-to-end paragraph recognition. It consists in a recurrent process during which the model focuses on specific parts of the input at each time step. To our knowledge, only two works focused on attention-based approaches.

In 2016, the authors of [82] proposed a CNN+MDLSTM model based on line-level attention *i. e.* that the model achieves a kind of implicit line segmentation. The encoder produces feature maps $\boldsymbol{f}$ from the input image while the attention module recurrently generates line representations $\boldsymbol{l^t}$ applying a weighted sum between the attention weights and the features. Finally, the decoder outputs character probabilities from this line representation. The model iterates a fixed number of times and all the probability lattices of each iteration are concatenated and aligned at paragraph-level with the standard CTC loss. The number

of iterations is chosen high enough to cover the maximum number of lines per paragraph in a given dataset.

The next year, T. Bluche *et. al.* proposed a second attention-based CNN+MDLSTM model in [83], based on the same encoder. The main difference relates to the attention mechanism which is at character level. It means that the model performs an implicit character segmentation: each iteration is dedicated to the prediction of one character. The model uses specific <start-of-paragraph> and <end-of-paragraph> tokens to initialize and stop the recurrent process, and is trained with the cross entropy loss. Both line-level and character-level models are represented in Figure 4.4.

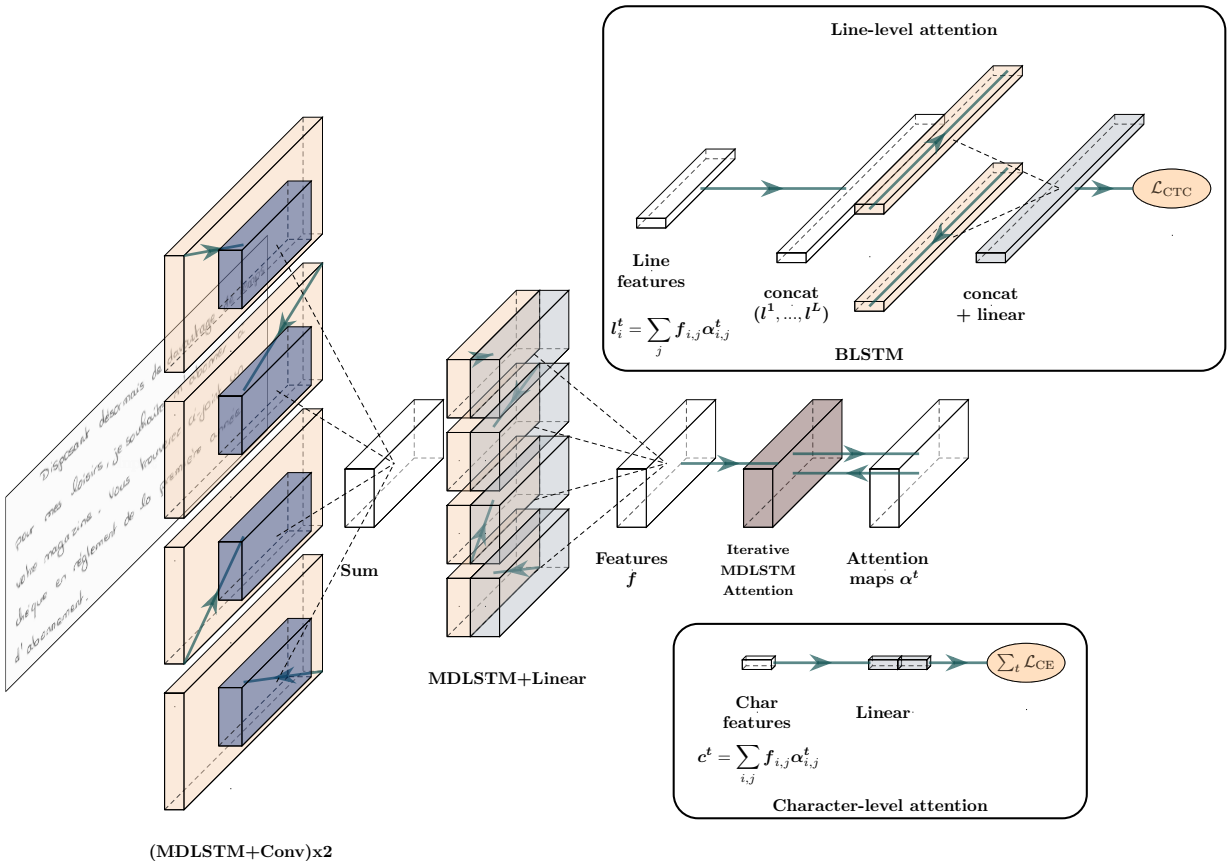


Figure 4.4: MDLSTM architecture with line-level and character-level attention.

While the one-shot approaches rely on a rule-based decoding process to generate the final prediction, the attention-based models are closer to the human way of reading: they learn the reading order. However, contrary to one-shot approaches, the recurrence is inherent to the attention-based models, theoretically leading to higher prediction times, especially with character-level attention. It led us to think that both approaches should be investigated. This is why we proposed two new architectures for end-to-end paragraph recognition:

- The Simple Predict & Align Network (SPAN). It is a one-shot approach based on an FCN model. Instead of unfolding the input image as in [144], we propose to train the proposed model to both predict and align characters so as to get vertical separation

between lines, preserving the two-dimensional nature of the task. In addition, while the approach proposed in [144] relies on dataset-specific hyperparameters for the dimension of the bi-linear interpolations, the SPAN is able to handle input images of variable sizes, making it flexible enough to be used on multiple datasets without modifying any hyperparameter.

- The Vertical Attention Network (VAN). It consists in an attention-based model which follows the same idea of implicit line segmentation as in [82]. However, we use an FCN encoder and an attention module without recurrent layers to reduce the computation time while implying few parameters at the same time. We also proposed a new vertical hybrid attention mechanism as well as a module dedicated to detect the end of the paragraph so as to stop the recurrent process.

4.3 SPAN: a Simple Predict & Align Network

We propose the SPAN as an end-to-end one-shot approach for the recognition of handwritten paragraphs. The SPAN follows a novel and easy approach for this task. Indeed, it is as simple as line-level HTR approaches as it relies on a recurrent-free process (we used an FCN architecture) and on the standard Connectionist Temporal Classification (CTC) loss. In the following, we present the SPAN architecture as well as the training strategy we use. We provide an experimental study and a discussion of the model. We provide all source code and pre-trained model weights at `https://github.com/FactoDeepLearning/SPAN`.

4.3.1 Architecture

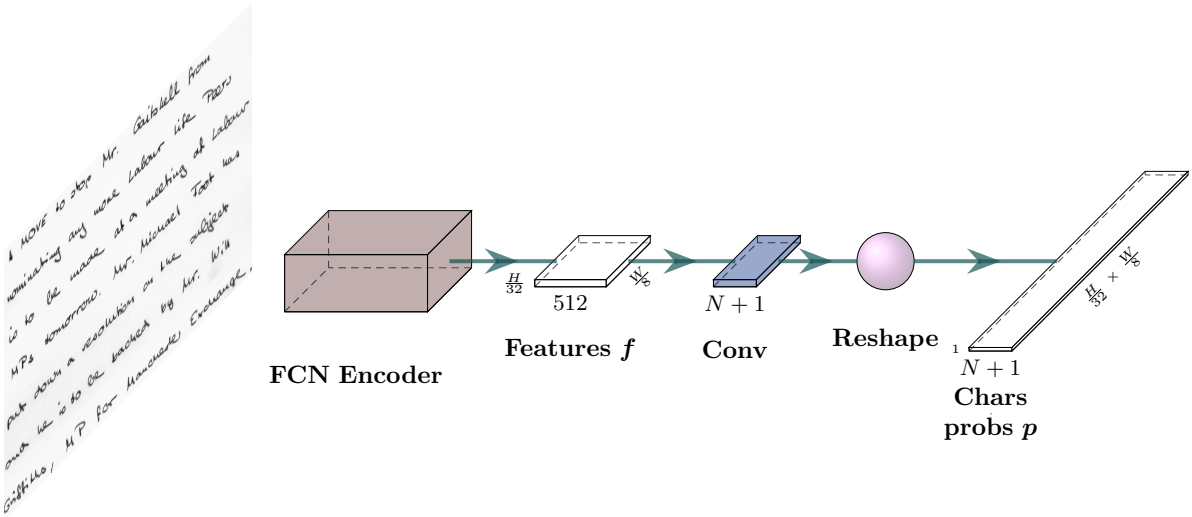


Figure 4.5: SPAN architecture overview.

The architecture of the proposed SPAN is depicted in Figure 4.5. We wanted to keep the original sizes of the input images in order to preserve both their ratio and their details

as well as to be flexible enough to adapt to a large variety of datasets. To this end, we use an FCN as the encoder to extract features from the 2D paragraph images. The decoder consists in a single convolutional layer to predict the characters probabilities. The idea is to predict the character probabilities from the extracted features while keeping the 2D nature of the task. It enables to have a global 2D context which is used to align the characters of a same text lines on the vertical dimension so as to implicitly segment the different text lines. The convolutional layer is followed by a row concatenation operation, to collapse the vertical dimension, reshaping the 2D latent space into a 1D latent space. As shown on Figure 4.6, the vertical prediction alignment enables to preserve the order of the text after the reshaping operation. This brings us back to a one-dimensional sequence alignment problem which is handled with the standard CTC loss.

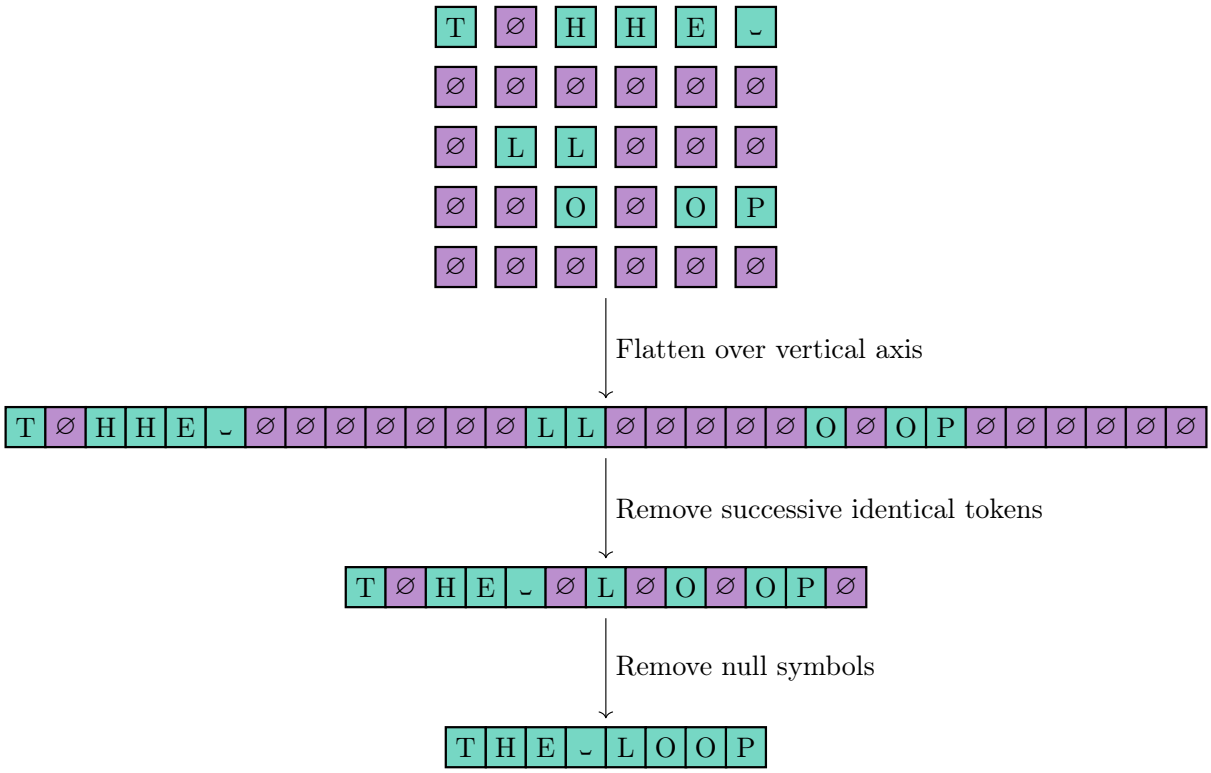


Figure 4.6: SPAN prediction and decoding process.

4.3.1.1 Encoder

The purpose of the encoder is to extract features from the input images. It is made up of some convolutions, with stride greater than 1×1, in order to reduce the memory consumption: it takes an input image $\boldsymbol{X} \in \mathbb{R}^{H \times W \times C}$ and outputs some feature maps $\boldsymbol{f} \in \mathbb{R}^{\frac{H}{32} \times \frac{W}{8} \times 512}$ where H, W and C are respectively the height, the width and the number of channels ($C = 1$ for a grayscale image, $C = 3$ for a RGB image). The encoder architecture is depicted in Figure 4.7.

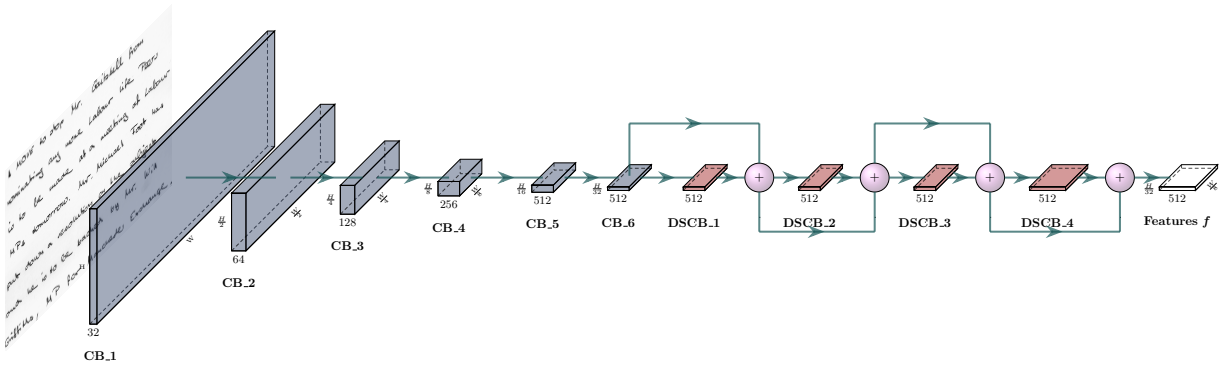


Figure 4.7: FCN Encoder overview. CB: Convolution Block, DSCB: Depthwise Separable Convolution Block.

It corresponds to an improved version of the Gated Fully Convolutional Network (GFCN) we proposed for line-level recognition. The main modifications are as follows. We introduce residual connections in order to have a better feature extraction by aggregating multi-scale information. We made the network simpler by using only two kind of convolution blocks, namely Convolution Blocks (CB) and Depthwise Convolution Blocks (DSCB). We introduce a new dropout strategy named Diffused Mix Dropout (DMD). We remove the gating component as we noticed that it was not beneficial for this new architecture. Stacking these convolutional blocks leads to a receptive field of 961 pixels high and 337 pixels wide.

Convolutional Block (CB) CB is defined as two convolutional layers followed by instance normalization and a third convolutional layer. Each convolutional layer uses 3×3 kernels and is followed by a ReLU activation function; zero padding is introduced to remove the kernel edge effect. While the first two convolutional layers have a 1×1 stride, the third one has a 1×1 stride for CB_1, 2×2 stride for CB_2 to CB_4 and 2×1 stride for CB_5 and CB_6. The strides are chosen to progressively decrease the dimensions. It is a trade-off between memory consumption and performance. This enables to divide the height by 32 and the width by 8. DMD is applied at three possible locations which are just after the activation layers of the three convolutional layers.

Depthwise Separable Convolution Block (DSCB) DSCB differs from CB in two respects. On the one hand, the standard convolutional layers are superseded by Depthwise Separable Convolutions (DSC) [41]. The aim is to reduce the number of parameters while keeping the same level of performance. On the other hand, the third convolutional layer has a fixed stride of 1×1. In this way, the shape is preserved until the last layer of the encoder. This enables to introduce residual connections with element-wise sum operator between the DSCB.

Diffused Mix Dropout Dropout is commonly used as regularization during training to avoid overfitting. We introduce Diffused Mix Dropout, a new dropout strategy that takes advantage of the two main modes proposed in the literature, namely standard (std.) dropout

[9] and spatial (or 2d) dropout [10]. We define the Mix Dropout (MD) layer as the layer which applies one of the two possible dropout modes, randomly. It enables to take advantage of the two implementations in a single layer. Diffused Mixed Dropout (DMD) consists in randomly applying MD at different locations among a set of pre-selected locations. In the model, we use DMD with dropout probability of 0.5 and 0.25 respectively for the standard and the 2d modes. Both modes have equivalent probabilities to be chosen at each execution. The benefit of using this dropout strategy is discussed in Section 4.4.5.3 through experiments on the VAN architecture.

4.3.1.2 Decoder

The decoder aims at predicting and aligning the probabilities of the characters and of the CTC blank label for each 2D position of the features $\boldsymbol{f}$. The decoder part is very simple as it only consists in a single convolutional layer followed by a flatten operation. The convolutional layer is performed with a 5×5 kernel, 1×1 stride and 2×2 padding. It outputs $N + 1$ channels, N being the size of the charset. Then, the $\frac{H}{32}$ rows are concatenated to obtain the one-dimensional prediction sequence $\boldsymbol{p} \in \mathbb{R}^{(\frac{H}{32} \cdot \frac{W}{8}) \times (N+1)}$ as depicted in Figure 4.8. The CTC loss is then computed between this one-dimensional prediction sequence and the paragraph transcription ground truth, without line breaks.

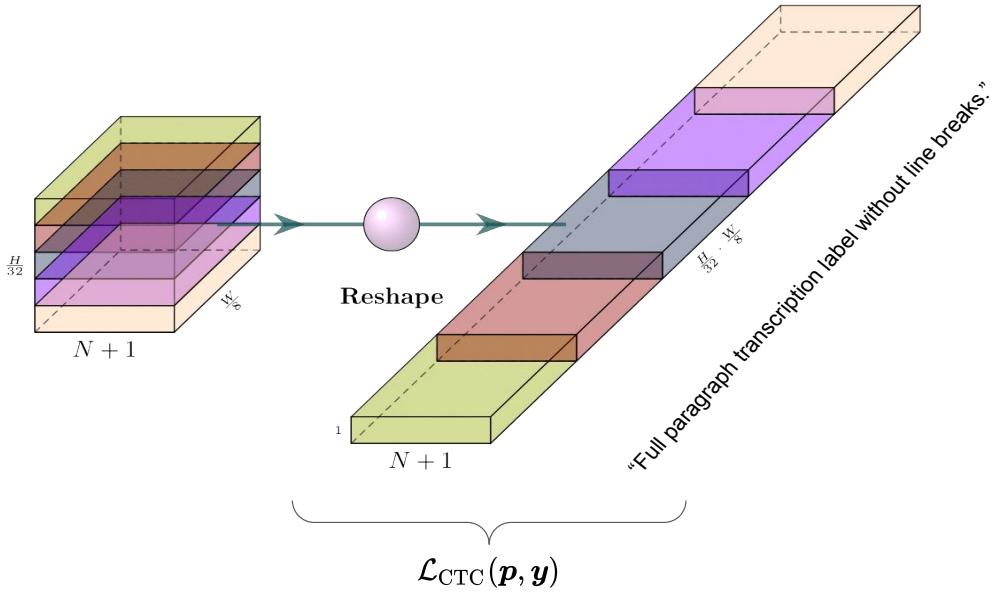


Figure 4.8: Reshaping operation and loss visualization. No computations are performed in the reshaping operation, both left and right tensors represent characters and CTC blank label probabilities. The CTC loss is computed between the one-dimensional probabilities sequence and the paragraph transcription.

We can highlight some important aspects about the decoder, that are illustrated in Figure 4.6 with an example of prediction for the target sequence "THE\nLOOP":

- In this work, the CTC blank label (or null symbol ∅) has a new function. In the standard CTC decoding process, the CTC blank label is used to recognize two identical successive characters and to predict "nothing", acting like a joker. Here, it is also used to separate lines in a two-dimensional context: predicting blank labels enables to fill the space between two successive text lines, as shown with the second row of prediction in Figure 4.6.

- One should notice that the prediction occurs before reshaping to 1D, which allows to take advantage of the two-dimensional context in the decision layer. This enables to localize the previous and next lines, and to align the predictions of the same text line on the same row *i.e.* to separate them from the predictions of the other text lines.

- Since the prediction rows are concatenated, they are processed sequentially; nothing prevents the model from predicting the beginning of the text line on one row and the end on the next one. This is only feasible if there is enough physical space between this text line and the following one so as not to overlap with the prediction of the next text line. In Section 4.3.3, we show that this enables the SPAN to process inclined lines.

4.3.2 Experimental conditions

We now describe the experimental conditions:

- Datasets: we evaluate the SPAN on three public datasets at paragraph level: RIMES 2011, IAM and READ 2016 (as described in Section 2.5.1).

- Pre-processings: paragraph images are downscaled by a factor of 2 through a bilinear interpolation leading to a resolution of 150 DPI. Gray-scaled images are converted into RGB images concatenating the same values three times, for transfer learning purposes. They are then normalized (zero mean and unit variance) considering the channels independently.

- Data Augmentation: a data augmentation strategy is applied at training time to reduce overfitting. The augmentation techniques are used in this order: resolution modification, perspective transformation, elastic distortion and random projective transformation (from [144]), dilation and erosion, brightness and contrast adjustment, and sign flipping. Each transformation has a probability of 0.2 to be applied. Except for perspective transformation, elastic distortion and random projective transformation which are mutually exclusive, each augmentation technique can be combined with the others.

- Pre-training: the weights of the SPAN encoder are initialized with those of a line-level HTR model trained on isolated text lines of the same dataset. This line-level HTR model is depicted in Figure 4.9. It is made up of the SPAN FCN encoder, followed by an Adaptive Max Pooling layer which pushes the vertical dimension to collapse. A final convolutional layer predicts the probability of the characters and of the CTC null symbol.

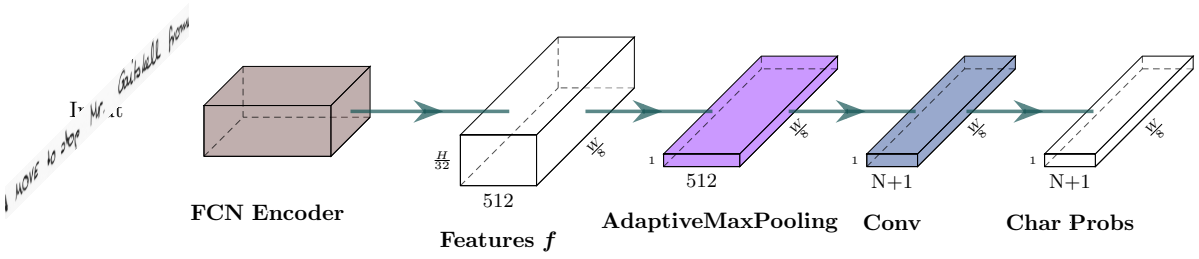


Figure 4.9: Line-level HTR model used for pre-training.

- Training details: we used the Pytorch framework to train and evaluate the models. In all experiments, the networks are trained with the Adam optimizer, with an initial learning rate of 10^{-4}. Trainings are performed on a single GPU Tesla V100 (32Gb), during 2 days, with a mini-batch size of 4 for paragraph images and 16 for text lines images. We used the original paragraph ground truth replacing line breaks by space characters.

- Metrics: we used the Character Error Rate (CER) and the Word Error Rate (WER) to evaluate the quality of the text recognition. They are both computed with the Levenshtein distance between the ground truth text and the predicted text at paragraph level, without line breaks. Those edit distances are then normalized by the length of the ground truth. For the WER, punctuation marks are not counted as words: they are considered as belonging to the previous word. Other metrics are provided in the following experiments such as the number of parameters implied by the models.

We do not use any post-processing *i.e.* we do not use any external language model nor lexicon constraint. Moreover, we only use best path decoding to get the final predictions from the character probabilities lattice. We use exactly the same training configuration from one dataset to another, without model modification, except for the last layer which depends on the charset size.

4.3.3 Experimental study

In the following, we compare the SPAN with the state-of-the-art approaches. We provide a visualization of the vertical alignment learned by the model and we study the impact of pre-training on the SPAN results.

4.3.3.1 Comparison with the state of the art

In this section, we compare our approach to state-of-the-art models on the RIMES 2011, IAM and READ 2016 datasets, at paragraph level and in the same conditions *i.e.* without external language model, external data nor lexicon constraint.

Prior to compare the obtained results to the state of the art, it is important to understand the experimental conditions of each of the methods. Table 4.1 shows model details

Table 4.1: Requirements comparison of the SPAN with the state-of-the-art approaches.

Architecture	# Param.	Max memory	Transcription label	Seg. label	PT	CL	HA
[102] RPN+CNN+BLSTM			Word	Word			
[104] RPN+CNN+BLSTM			Word	Word			
[107] RPN+CNN+BLSTM			Line	Line			
[83] CNN+MDLSTM**			Paragraph	Paragraph	Line	✓	✗
[82] CNN+MDLSTM*			Paragraph	Paragraph	Line	✓	✗
[144] GFCN	16.4 M	8.8 Gb	Paragraph	Paragraph	✗	✗	✓
[145] SPAN (FCN)	19.2 M	5.1 Gb	Paragraph	Paragraph	Line	✗	✗

* With line-level attention.
** with character-level attention.

that should be taken into account to fairly compare the following tables of results. Quantitative metrics are computed for the IAM dataset, without automatic mixed precision (for a fair comparison with respect to the memory usage). From left to right, the columns respectively denote the architecture, the number of trainable parameters, the maximum GPU memory usage during training (for a mini-batch size of 1, data augmentation included), the minimum transcription level required, the minimum segmentation level required, the use of Pre-Training (PT) on sub-images, the use of specific Curriculum Learning (CL) and finally the Hyperparameter Adaptation (HA) requirements from one dataset to another.

As one can see, models from [102, 104, 107] require transcription and segmentation labels at word or line levels to be trained, which implies more costly annotations. The models from [82, 83] and the SPAN are pre-trained on text line images to speed up convergence and to reach better results, thus also using line segmentation and transcription labels even if it is not strictly necessary. [82, 83] used a specific curriculum learning method for training. In [144], some hyperparameters must be modified from one dataset to another in order to reach optimal performance, namely the fixed input dimension and two intermediate upsampling sizes which are crucial. We do not have such problem since we are working with input images of variable size and we focus on the resolution to be robust to the variety of datasets. Moreover, despite a larger number of parameters (+ 17% compared to [144]), the SPAN requires less GPU memory which is a critical point when training deep neural networks.

Table 4.2 shows the results of the SPAN compared to the state-of-the-art approaches, for the RIMES 2011, IAM and READ 2016 datasets respectively. To our knowledge, there are no results reported in the literature on the READ 2016 dataset at paragraph level. One can notice that we reach competitive results on those three datasets, each having its own complexities, with an architecture as simple as an FCN. In addition, we used exactly the same hyperparameters from one dataset to another, showing the robustness of this approach.

Results include model pre-training on text line images but the model can be trained without pre-training *i.e.* without using any line-level annotation, while keeping competitive results, as shown in Section 4.3.3.3.

4.3.3.2 SPAN prediction visualization

Figure 4.10 presents a visualization of the SPAN prediction for an example of the RIMES 2011 test set. Character predictions are shown in red; they seem like rectangle since they are

Table 4.2: Comparison of the SPAN results with the state-of-the-art approaches on the RIMES 2011, IAM and READ 2016 datasets.

Architecture	CER (%) validation	WER (%) validation	CER (%) test	WER (%) test
RIMES 2011				
[82] CNN+MDLSTM*	2.5	12.0	2.9	12.6
[107] RPN+CNN+BLSTM			**2.1**	9.3
[145] SPAN (FCN)	3.56	14.29	4.17	15.61
IAM				
[102] RPN+CNN+BLSTM*	13.8		15.6	
[104] RPN+CNN+BLSTM			8.5	
[107] RPN+CNN+BLSTM			6.4	23.2
[83] CNN+MDLSTM**			16.2	
[82] CNN+MDLSTM*	4.9	17.1	7.9	24.6
[144] GFCN			**4.7**	
[145] SPAN (FCN)	3.57	15.31	5.45	19.83
READ 2016				
[145] SPAN (FCN)	5.09	23.69	**6.20**	25.69

* with line-level attention.
** with character-level attention.

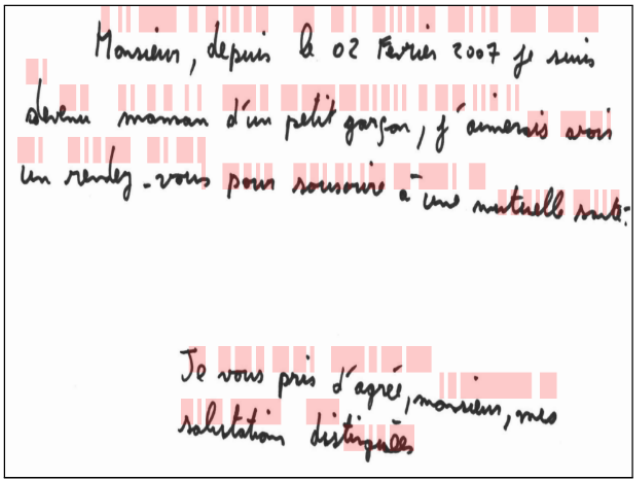
resized to fit the input image size (the features size is $\frac{H}{32} \times \frac{W}{8}$). Combined with the receptive field effect, this explains the shift that can occur between the prediction and the text. As one can notice, text line predictions are totally aligned, or aligned by blocks; the lines are well separated by blank labels, which act as line spacing labels. As one can see, this block alignment enables to handle downward inclined lines, especially for lines 3 and 4. Moreover, the model does not degrade in the presence of large line spacing.

4.3.3.3 Impact of pre-training

In this experiment, we try to highlight the impact of pre-training on the SPAN results. To this end, we compare two pre-training methods at line level: one focusing only on the optical Recognition task (R) and the second one focusing on both Recognition and prediction Alignment (R&A). Let's define the following pre-training (PT) approaches:

- SPAN-Line-R&A: the SPAN is trained with line-level images. Here, the network has to learn both the recognition and the alignment tasks. Indeed, there can be multiple rows of features for a single text line. This way, the model must learn the alignment aspect, in addition to the character recognition, to achieve correct predictions.
- Pool-Line-R: it corresponds to the pre-training strategy presented in Section 4.3.2. The line-level HTR model depicted in Figure 4.9 is trained on isolated text line images to only focus on the recognition task. Since the vertical dimension is collapsed (through adaptive max pooling), the network does not need to care about the vertical alignment of the predictions.

It leads to three different training approaches:



Monsieur, depuis la 02 Ferries 2007 je suis
de
enu maman d'un petit garçon, j'aimer
ais avoir
un rendez-vous
s pour sousoire à une m
ntuelle sont

Je vous pris d'agrée, m
ansieurs, mes
salutation dis
tinguées

Monsieur, depuis l**a** 02 Fe**r**rie**s** 2007 je suis de***v***enu maman d'un petit garçon, j'aimerais avoir un rendez-vous**s** pour sous**o**ire à une m**n**tuelle s**o**nt***é.*** Je vous pris d'agrée, m**a**nsieurs, mes salutation***s*** distinguées

Figure 4.10: SPAN predictions visualization for a RIMES 2011 test example. Left: 2D characters predictions are projected on the input image. Red color indicates a character prediction while transparency means blank label prediction. Right: row by row text prediction. Bottom: full text prediction where errors are shown in bold and missing letters are shown in italic.

- SPAN-Scratch: the SPAN is trained directly on paragraph images without pre-training.
- SPAN-PT-R: SPAN weights are initialized with Pool-Line-R ones. It is then trained with paragraph images.
- SPAN-PT-R&A: SPAN weights are initialized with SPAN-Line-R&A ones. It is then trained with paragraph images.

One can note that the vertical receptive field is larger than the height of the images of isolated text lines. Thus, when the model switches from line images to paragraph images, the decision benefits from more context, which replaces part of the previously used padding.

Results are given in Table 4.3. Focusing on the line-level section (pre-training step), one can notice that, as expected, we reached better results on text lines when the task is reduced to optical recognition compared to the task of recognition and alignment, whatever the dataset. This leads to a CER improvement of 0.94 point for IAM, 0.79 point for RIMES 2011 and 0.38 point for READ 2016. This can be explained by the difficulty of the task: it is easier to just recognize characters than to recognize characters and align their predictions on the vertical axis. In addition, we assume that aligning the predictions does not help the character recognition.

Now, comparing the paragraph-level approaches, one can notice that, except for the RIMES 2011 CER, pre-training leads to better results, and sometimes by far (-2.93 points of CER for READ 2016). Moreover, pre-training on an easier task, *i.e.* only on the optical recognition, is even more efficient. We assume that focusing only on the recognition part enables to improve the feature extraction part compared to both recognizing and aligning the predictions.

Table 4.3: Impact of pre-training the SPAN on line images for the IAM, RIMES 2011 and READ 2016 datasets. Results are given on the test sets.

Approach	IAM		RIMES 2011		READ 2016	
	CER (%)	WER (%)	CER (%)	WER (%)	CER (%)	WER (%)
Line-level (pre-training)						
Pool-Line-R	**4.82**	**18.17**	**3.02**	**10.73**	**4.56**	**21.07**
SPAN-Line-R&A	5.76	21.33	3.81	13.80	4.94	22.19
Paragraph-level (training)						
SPAN-Scratch	6.46	23.75	**4.15**	16.31	9.13	36.63
SPAN-PT-R	**5.45**	**19.83**	4.74	**15.55**	**6.20**	**25.69**
SPAN-PT-R&A	5.78	21.16	4.17	15.71	6.62	27.38

4.3.4 Discussion

We propose the Simple Predict & Align Network (SPAN) as an end-to-end recurrence-free FCN model performing HTR at paragraph level. It reaches competitive results on the RIMES 2011, IAM and READ 2016 datasets without any architecture or training adaptation from one dataset to another. It shows that one can achieve HTR at paragraph level with a standard FCN network and using the standard CTC loss provided that the architecture includes a way to serialize the representation, through a simple flattening operation for instance.

The proposed architecture has other advantages. It only needs transcription label at paragraph level without line breaks, leveraging the need for handmade annotation, which is a critical point for a deep learning system. It can handle input images of variable sizes, making it robust enough to adapt to multiple datasets; it is also able, through the reshaping operation, to deal with downward inclined text lines. However, this same reshaping operation prevent the model from handling upward ones due to the fixed reshaping order. The use of a second SPAN model whose reshaping operation would have a reversed ordering method could enable to deal with both upward and downward inclined lines.

Moreover, since we are using the standard CTC loss, one can easily add standard character or word language model to further improve the results. The SPAN could also adapt to full page documents since it is based on the image resolution regardless of its size. However, it has to be noticed that this model is limited to single-column multi-line text images due to the row concatenation operation.

We now present the second paragraph-level architecture we propose: the VAN.

4.4 VAN: a Vertical Attention Network

We propose the Vertical Attention Network as an end-to-end attention-based approach for the recognition of handwritten paragraphs. It is based on the idea of a recurrent process which successively focuses on the different text lines, one after the other, and which learns to detect the end of the paragraph in order to stop the prediction. This line attention enables to remain in a one-dimensional sequence alignment problem between the recognized text lines and their ground truth transcriptions, which is handled by the CTC.

In the following, we present the VAN architecture and the training strategy. We provide an experimental study, a visualization of the attention process, as well as a discussion of the model. We provide all source code and pre-trained model weights at `https://github.com/FactoDeepLearning/VerticalAttentionOCR`.

4.4.1 Architecture

The VAN follows the encoder-decoder principles with an attention module. It takes as input an image $\boldsymbol{X}$ and recurrently recognizes the L text lines $[\boldsymbol{y^1}, ..., \boldsymbol{y^L}]$ present in it. Each text line $\boldsymbol{y^t}$ is a sequence of tokens from an alphabet $\mathcal{A}$, with $|\mathcal{A}| = N$.

The overall model is presented in Figure 4.11. It first extracts features $\boldsymbol{f}$ from $\boldsymbol{X}$. We wanted the encoder stage to be modular enough in order to be plugged in different architectures dedicated to text lines, paragraphs or documents recognition without needing any adaptation. To this end, we chose an FCN encoder that can deal with input images of variable heights and widths, possibly containing multiple lines of text. The attention module is the main control block: it recurrently produces vertical attention weights that focus where to select the next features for the recognition of the next text line. This way, it recurrently generates as many text line features/representations $\boldsymbol{l^t}$ as there are text lines in the input image. It also detects the end of the paragraph leading to the end of the whole process. Finally, the decoder stage produces character probabilities $\boldsymbol{p^t}$ for each frame of each generated line features, and the best path decoding strategy is used. The model is trained using a combination of two losses: the CTC loss for the recognition task, through the line-by-line alignment between recognized text lines (through its probabilities lattice $\boldsymbol{p^t}$) and ground truth line transcriptions $\boldsymbol{y^t}$, and the Cross Entropy (CE) loss for the end-of-paragraph detection. We now describe each module in detail.

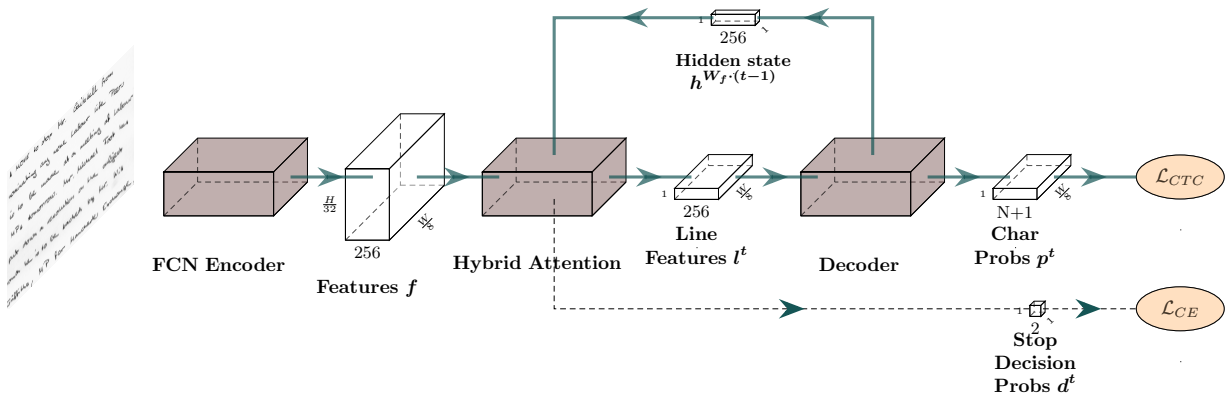


Figure 4.11: VAN architecture overview.

4.4.1.1 Encoder

An FCN encoder is used to extract features from the input paragraph images. It takes as input an image $\boldsymbol{X} \in \mathbb{R}^{H \times W \times C}$, with H, W and C being respectively the height, the width and

the number of channels. Then, it outputs some feature maps for the whole paragraph image: $\boldsymbol{f} \in \mathbb{R}^{H_f \times W_f \times C_f}$ with $H_f = \frac{H}{32}$, $W_f = \frac{W}{8}$ and $C_f = 256$. Those features must contain enough information to recognize the characters afterwards, while preserving the two-dimensional nature of the task.

We used the SPAN encoder as it shows interesting results for the same task. The only difference is the maximum number of channels which is set to 256 instead of 512. It considerably reduces the number of trainable parameters at stake, from 18.1M to 1.7M, without impacting the performance for the VAN architecture.

4.4.1.2 Attention

At this step, we have computed the features $\boldsymbol{f}$. This is a static representation that preserves the two-dimensional aspect of the task: it never changes through the attention process. The purpose of the attention module is to recursively produce text line representations, in the desired reading order, from top to bottom for example. In addition, it has to decide when to stop generating a new line representation, *i.e.* to detect the end of the paragraph. The whole process is depicted in Figure 4.12.

Line features generation

Given an input image with L text lines, the attention module successively produces L line features. In the following, the process of the t^{th} line will be referred as attention step t. To produce the successive line features, we chose to use a soft attention mechanism, as proposed by Bahdanau in [58], and more specifically a hybrid attention. The soft attention is a way to focus on specific frames among a one-dimensional sequence of frames. It involves the computation of attention weights $\boldsymbol{\alpha_i^t}$ for each frame i at attention step t. These attention weights sum to 1 and quantify the importance of each frame for the attention step t. We apply the attention on the vertical axis only, in order to focus on specific features row at each attention step t. This way, we need one weight per features row: we roughly expect one weight to be near 1, selecting the correct feature row and the others to be near 0. Each attention step t is dedicated to process the text line number t. The corresponding line features $\boldsymbol{l^t}$ can be computed as a weighted sum between the feature rows and the attention weights:

$$\boldsymbol{l^t} = \sum_{i=1}^{H_f} \boldsymbol{\alpha_i^t} \cdot \boldsymbol{f}_i. \tag{4.1}$$

As one can note, this weighted sum enables to provide line features $\boldsymbol{l^t}$ to be a one-dimensional sequence and thus to use the standard CTC loss for each recognized text line.

The hybrid aspect means that attention weights are computed from both the content and the location output layers. In our case, the attention weights computation combines three elements:

- $\boldsymbol{f} \in \mathbb{R}^{H_f \times W_f \times C_f}$: the features in which we want to select specific line features.
- $\boldsymbol{\alpha^{t-1}} \in \mathbb{R}^{H_f}$: the previous attention weights. To know which line to select, we must know which ones have already been processed. This is the location-based part.

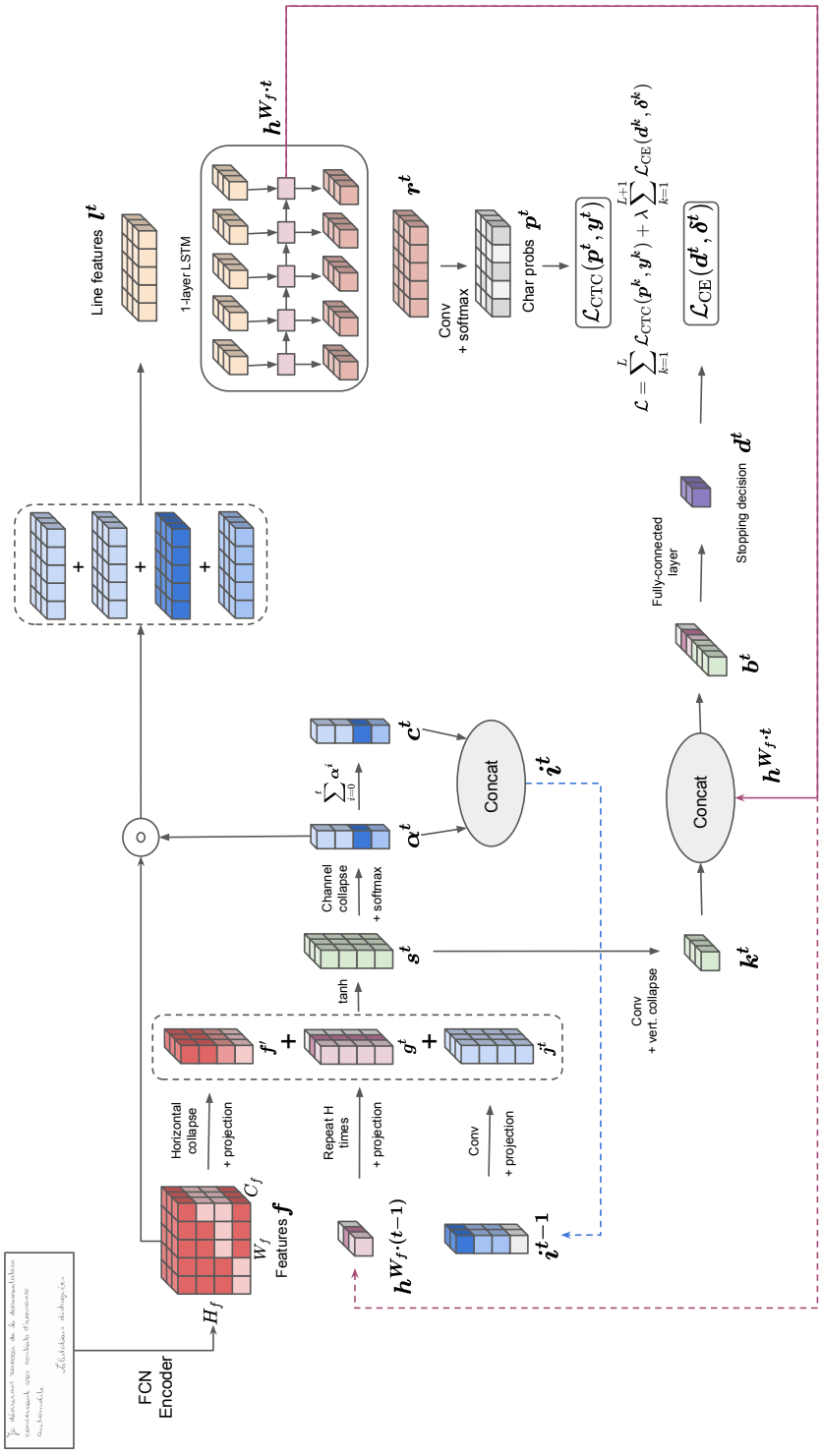


Figure 4.12: Focus on the VAN hybrid attention mechanism.

- $\boldsymbol{h^{W_f \cdot (t-1)}} \in \mathbb{R}^{C_h}$: the decoder hidden state after processing the previous text line (each line features is made up of W_f frames). It contains information about what has been recognized so far. This is the content-based part.

$\boldsymbol{f}$, $\boldsymbol{\alpha^{t-1}}$ and $\boldsymbol{h^{W_f \cdot (t-1)}}$ cannot be combined directly due to dimension mismatching. Since we want to normalize the attention weights over the vertical axis, we must collapse the horizontal axis of the features. The idea is that we only need to know if there is at least one character in a features row to decide to process it as a text line. Thus, we can collapse this horizontal dimension without information loss. This collapse is carried out in two steps: we first get back to a fixed width of 100 (since inputs are of variable sizes) through Adaptive Max Pooling. Then, a densely connected layer pushes the horizontal dimension to collapse. The remaining vertical representation is called $\boldsymbol{f'} \in \mathbb{R}^{H_f \times C_f}$.

Instead of using directly $\boldsymbol{\alpha^{t-1}}$, we found that combining it with a coverage vector $\boldsymbol{c^t} \in \mathbb{R}^{H_f}$ is more beneficial. $\boldsymbol{c^t}$ is defined as the sum of all previous attention weights:

$$\boldsymbol{c^t} = \sum_{k=0}^{t} \boldsymbol{\alpha^k}. \tag{4.2}$$

$\boldsymbol{c^t}$ enables to keep track of all previous line positions processed. $\boldsymbol{c^t}$ is clamped between 0 and 1 for stability; we only need to know whether a row has already been processed or not. Bigger values would not have sense. We combined $\boldsymbol{c^t}$ and $\boldsymbol{\alpha^t}$ through concatenation over the channel axis, leading to $\boldsymbol{i^t} \in \mathbb{R}^{H_f \times 2}$. Finally, we extract some context information from $\boldsymbol{i^t}$ through a one-dimensional convolutional layer with $C_j = 16$ filters of kernel size 15 and stride 1 with zero padding, followed by instance normalization. This outputs $\boldsymbol{j^t} \in \mathbb{R}^{H_f \times C_j}$ which contains all the contextualized spatial information from past attention steps.

We can now compute the attention weights $\boldsymbol{\alpha^t}$ as follows:

- For each row i, we compute the associated multi-scale information $\boldsymbol{s_i^t}$, gathering all the elements we have seen previously:

$$\boldsymbol{s_i^t} = \tanh(\boldsymbol{W_f} \cdot \boldsymbol{f_i'} + \boldsymbol{W_j} \cdot \boldsymbol{j_i^t} + \boldsymbol{W_h} \cdot \boldsymbol{h^{W_f \cdot (t-1)}}). \tag{4.3}$$

  Indeed, $\boldsymbol{s_i^t}$ contains both local and global information: information from features and previous attention weights can be considered as local since they are position-dependent; and information from the decoder hidden state can be seen as global since it is related to all characters recognized so far. $\boldsymbol{W_f}$, $\boldsymbol{W_j}$ and $\boldsymbol{W_h}$ are weights of densely connected layers unifying the number of channels of the elements to be summed to $C_u = 256$ ($\boldsymbol{W_f} \in \mathbb{R}^{C_f \times C_u}$, $\boldsymbol{W_j} \in \mathbb{R}^{C_j \times C_u}$ and $\boldsymbol{W_h} \in \mathbb{R}^{C_h \times C_u}$).

- Score $\boldsymbol{e_i^t}$ are then computed for each row i:

$$\boldsymbol{e_i^t} = \boldsymbol{W_a} \cdot \boldsymbol{s_i^t}, \tag{4.4}$$

  where $\boldsymbol{W_a}$ are weights of a densely connected layer reducing the channel axis to get a single value ($\boldsymbol{W_a} \in \mathbb{R}^{C_u \times 1}$).

- Attention weights are finally computed through softmax activation applied over these scores:

$$\boldsymbol{\alpha}_i^{\boldsymbol{t}} = \frac{\exp(\boldsymbol{e}_i^{\boldsymbol{t}})}{\sum_{k=1}^{H_f} \exp(\boldsymbol{e}_k^{\boldsymbol{t}})}. \tag{4.5}$$

We emphasize that, although the attention module produces a vertical focus $\boldsymbol{l}^{\boldsymbol{t}}$, the line recognizer has a broad view of the input signal due to the size of the receptive field. Therefore, it makes the method robust to inclined or non-straight lines.

End-of-paragraph detection

One of the main intrinsic problems of paragraph recognition is the unknown number of text lines to be recognized. As the text recognition is processed sequentially, an end-of-paragraph detection is needed. To solve this problem, we compare three different approaches we named *fixed-stop approach*, *early-stop approach* and *learned-stop approach*.

For each of these approaches, we used the CTC loss to align the line prediction (probabilities lattice $\boldsymbol{p}^{\boldsymbol{t}}$ of length W_f), with the corresponding line transcription $\boldsymbol{y}^{\boldsymbol{t}}$. It involves the addition of the null symbol (also known as blank), leading to a new alphabet $\mathcal{A}' = \mathcal{A} \cup \{blank\}$.

We also defined a stopping criterion at evaluation time to avoid infinite loops. It consists in a constant $l_{\max}$ large enough to cover the biggest paragraph of the dataset. This number can easily be set to match the datasets involved; in our case $l_{\max} = 30$.

The *fixed-stop approach* is the simplest way to handle the end-of-paragraph detection issue. The model iterates $l_{\max}$ times and stops, as proposed in [82]. Additional fictive lines $[\boldsymbol{y}^{\boldsymbol{L+1}}, ..., \boldsymbol{y}^{\boldsymbol{l_{max}}}]$ are added to the ground truth as empty strings. The idea is that the extra iterations will focus their attention on interlines, only predicting null symbols, to avoid recognizing the same line multiple times. During training the loss is defined as follows:

$$\mathcal{L}_{\text{fs}} = \sum_{k=1}^{l_{\max}} \mathcal{L}_{\text{CTC}}(\boldsymbol{p}^{\boldsymbol{k}}, \boldsymbol{y}^{\boldsymbol{k}}). \tag{4.6}$$

This approach leads to an additional processing cost during training and evaluation due to the extra iterations needed.

To alleviate this issue, we propose the *early-stop approach*. The idea is that we can consider that the lines have all been predicted as soon as the current prediction is an empty line (only null symbols predicted). This way, we only have to add one additional line to the ground truth: y_{L+1}. The loss is then defined this way:

$$\mathcal{L}_{\text{es}} = \sum_{k=1}^{L+1} \mathcal{L}_{\text{CTC}}(\boldsymbol{p}^{\boldsymbol{k}}, \boldsymbol{y}^{\boldsymbol{k}}). \tag{4.7}$$

Finally, we propose the *learned-stop approach* as a more elegant way to solve this problem. It consists in learning when to stop recognizing a new text line *i.e.* to detect when the whole paragraph has been processed. This end-of-paragraph detection is performed at each iteration, computing the probability $\boldsymbol{d}^{\boldsymbol{t}}$ to stop or to continue the recognition. More

specifically, $\boldsymbol{d^t}$ determines whether $\boldsymbol{p^t}$ should be considered or not. This way, the model iterates $L+1$ times to learn to predict the end of the process.

We decided to compute this probability from two elements: the multi-scale information $\boldsymbol{s^t}$ which brings some visual information about what has already been processed and what remains to be decoded, and the decoder hidden state $\boldsymbol{h^{W_f \cdot t}}$ which can contain information about what has already been recognized. Indeed, it is more likely to be the end of the paragraph if the last recognized character is a dot for example.

To this end, we first have to collapse the vertical dimension of $\boldsymbol{s^t}$ for dimension matching purposes. This is carried out in the following way:

- A one-dimensional convolutional layer with kernel size 5, stride 1 and zero padding is applied on $\boldsymbol{s^t}$.
- Adaptive Max Pooling is used to reduce the height to a fixed value of 15 (since input are of variable sizes).
- A densely connected layer pushes the vertical dimension to collapse, leading to $\boldsymbol{k^t} \in \mathbb{R}^{C_f}$.

Then, the produced tensor $\boldsymbol{k^t}$ is combined with $\boldsymbol{h^{W_f \cdot t}}$ through concatenation over the channel axis, leading to $\boldsymbol{b^t}$.

Finally, the dimension is reduced to 2 in order to compute the probabilities $\boldsymbol{d^t} \in \mathbb{R}^2$ through a densely connected layer of weights $\boldsymbol{W}_d \in \mathbb{R}^{(C_f+C_h)\times 2}$:

$$\boldsymbol{d^t} = \boldsymbol{W}_d \cdot \boldsymbol{b^t}. \tag{4.8}$$

This approach leads to the addition of a cross-entropy loss to the CTC loss, applied to the decision probabilities $\boldsymbol{d^t}$. The corresponding ground truth $\boldsymbol{\delta^t}$ is one-hot encoded, deduced from the line breaks. The cross-entropy loss is defined as follows:

$$\mathcal{L}_{\text{CE}}(\boldsymbol{d^t}, \boldsymbol{\delta^t}) = -\sum_{i=1}^{2} \boldsymbol{\delta}_i^t \log \boldsymbol{d}_i^t. \tag{4.9}$$

The final loss is then:

$$\mathcal{L}_{\text{ls}} = \sum_{k=1}^{L} \mathcal{L}_{\text{CTC}}(\boldsymbol{p^k}, \boldsymbol{y^k}) + \lambda \sum_{k=1}^{L+1} \mathcal{L}_{\text{CE}}(\boldsymbol{d^k}, \boldsymbol{\delta^k}), \tag{4.10}$$

where λ is set to 1.

These three approaches are compared through experiments presented in Section 4.4.3.3.

4.4.1.3 Decoder

The decoder aims at recognizing a sequence of characters from the current line features $\boldsymbol{l^t}$, *i.e.* a whole text line. To this end, we can process $\boldsymbol{l^t}$ as we do in standard line-level HTR approaches, since the vertical axis is already collapsed: $\boldsymbol{l^t}$ is a one-dimensional sequence of features. First, we apply a single Long-Short Term Memory (LSTM) layer with $C_h = 256$

cells that outputs another representation of same dimension $\boldsymbol{r^t}$ which includes some context due to the recurrence over the horizontal axis. The LSTM hidden states $\boldsymbol{h^0}$ are initialized with zeros for the first line; they are maintained from one line to the next one to take advantage of the context at paragraph level. Then, a one-dimensional convolutional layer with kernel 1 going from 256 to $N + 1$ channels is applied in order to produce $\boldsymbol{p^t}$, the *a posteriori* probabilities of each character and the CTC null symbol, for each of the W_f frames. N is the size of the character set. Best path decoding is used to get the final characters sequence. Successive identical characters and CTC null symbols are removed through the CTC decoding algorithm, leading to the final text. All the recognized text lines are concatenated with a whitespace character as separator to get the whole paragraph transcription p_{pg}.

4.4.2 Experimental conditions

The experimental conditions are nearly identical to those used to train the SPAN: same datasets (RIMES 2011, IAM, READ 2016), same data augmentation strategy, same post-processing (best path decoding, no external data or language model) and same line-level pre-training strategy if not stated otherwise. The differences are as follows:

- We used an additional pre-processing step for the VAN, we zero pad the input images to reach a minimum height of 480 px and a minimum width of 800 px when necessary. This assures that the minimum features width will be 100 and the minimum features height will be 15, which is required by the model as described previously.
- We considered punctuation characters as words for the WER computation, as in the READ 2016 competition [3].

In addition, we propose a comparison with a three-step approach in the following. This way, in addition to the CER and to the WER, we used two other metrics to evaluate the segmentation stage: the Intersection over Union (IoU) and the mean Average Precision (mAP) (detailed in Section 2.5.2.1). The segmentation is applied at pixel level with two classes, namely text and background. We compute the global IoU over a set of images by weighting the image IoU by its number of pixels. We compute the mAP for an image as the average of AP computed for IoU thresholds between 50% and 95% with a step of 5%. Image mAPs are weighted by the number of pixels of the images to give the global mAP of a set.

Other metrics such as the number of parameters, the training time or the prediction time are useful to compare models. The training time is computed as the time to reach 90% of the convergence. This is a more relevant value since tiny fluctuations can occur after numerous epochs.

Training details

We used the Pytorch framework with the apex package to enable automatic mixed precision training thus reducing the memory consumption. We used the Adam optimizer for all experiments with an initial learning rate of 10^{-4}. Trainings are performed during two days on

a single GPU Tesla V100 (32Gb). Models have been trained with mini-batch size of 16 for line-level HTR model and mini-batch size of 8 for the segmentation model and for the VAN.

We use exactly the same hyperparameters from one dataset to another. Moreover, the model architecture is the same for each dataset: the last layer is the only difference since the datasets do not have the same character set size.

We used the line break annotations to split the ground truth into a sequence of text line transcriptions. This enables us to use the CTC loss to train the VAN through a line-by-line alignment between the recognized text lines and the ground truth line transcriptions.

Learned-stop training and prediction processes are respectively detailed in Algorithm 2 and 3. The algorithms for the fixed-stop and early-stop approaches are similar. All the instructions related to the variables σ_{CE}, $\boldsymbol{d^t}$ and $\boldsymbol{\delta^t}$ must be removed. Also, the for loop iterates $l_{\max}$ times instead of $L+1$ times for the fixed-stop approach.

Algorithm 2: VAN training process.

input: paragraph image $\boldsymbol{X}$, ground truth transcription $\boldsymbol{y}$ composed of L lines $[\boldsymbol{y^1}, ..., \boldsymbol{y^L}]$

$\boldsymbol{\alpha^0} = \boldsymbol{0}$, $\boldsymbol{h^0} = \boldsymbol{0}$, $\sigma_{\text{CTC}} = 0$, $\sigma_{\text{CE}} = 0$; `// Initialization`

$\boldsymbol{f} = \text{Encoder}(\boldsymbol{X})$; `// Extract 2d features` $\boldsymbol{f}$

`// For each line of the paragraph`

for *t=1* **to** *L+1* **do**

  `/* Compute attention weights` $\boldsymbol{\alpha^t}$ `for each features row to generate 1d line features` $\boldsymbol{l^t}$`. Also compute end-of-paragraph probabilities` $\boldsymbol{d^t}$ `*/`

  $\boldsymbol{l^t}, \boldsymbol{\alpha^t}, \boldsymbol{d^t} = \text{Attention}(\boldsymbol{f}, \boldsymbol{\alpha^{t-1}}, \boldsymbol{h^{W_f \cdot (t-1)}})$;

  `/* Determine ground truth for end-of-paragraph detection task */`

  if $t < L+1$ **then**

    `/* Compute character and CTC null symbol probabilities` $\boldsymbol{p^t}$ `for each frame of` $\boldsymbol{l^t}$ `*/`

    $\boldsymbol{p^t}, \boldsymbol{h^{W_f \cdot t}} = \text{Decoder}(\boldsymbol{l^t}, \boldsymbol{h^{W_f \cdot (t-1)}})$;

    $\sigma_{\text{CTC}} \mathrel{+}= \mathcal{L}_{\text{CTC}}(\boldsymbol{p^t}, \boldsymbol{y^t})$; `// Compute CTC loss`

    $\boldsymbol{\delta^t} = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$;

  else

    $\boldsymbol{\delta^t} = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$;

  $\sigma_{\text{CE}} \mathrel{+}= \mathcal{L}_{\text{CE}}(\boldsymbol{d^t}, \boldsymbol{\delta^t})$; `// Compute cross-entropy loss`

backward($\sigma_{\text{CTC}} + \sigma_{\text{CE}}$); `// Backpropagation`

We can summarize those processes as follows. First, the input images are pre-processed and augmented (at training time only) as described previously. Then, the encoder extracts features $\boldsymbol{f}$ from them. The attention module recurrently generates line features $\boldsymbol{l^t}$ until $l_{\max}$ is reached or, for the learned-stop approach, until $\boldsymbol{d^t}$ probabilities are in favor of stopping the process. The decoder outputs character probabilities from each line features $\boldsymbol{l^t}$, which are

Algorithm 3: VAN prediction process.

input: paragraph image $\boldsymbol{X}$

$\boldsymbol{\alpha^0} = \mathbf{0}$, $\boldsymbol{h^0} = \mathbf{0}$, $\boldsymbol{d^1} = \begin{pmatrix}0\\1\end{pmatrix}$, $l_{\max} = 30$, $t = 1$, $p_{\text{pg}} =$“ ” ;

$\boldsymbol{f} = \text{Encoder}(\boldsymbol{X})$;

while $t \leq l_{\max}$ & $\text{argmax}(\boldsymbol{d^t}) == 1$ **do**

  $\boldsymbol{l^t}, \boldsymbol{\alpha^t}, \boldsymbol{d^t} = \text{Attention}(\boldsymbol{f}, \boldsymbol{\alpha^{t-1}}, \boldsymbol{h^{W_f \cdot (t-1)}})$;

  if $\text{argmax}(\boldsymbol{d^t}) == 1$ **then**

    $\boldsymbol{p^t}, \boldsymbol{h^{W_f \cdot t}} = \text{Decoder}(\boldsymbol{l^t}, \boldsymbol{h^{W_f \cdot (t-1)}})$;

    $p_{\text{pg}} = \text{concatenate}(p_{\text{pg}},\text{“ ”}, \text{ctc\_decoding}(\boldsymbol{p^t}))$;

  $t = t + 1$;

then decoded through the CTC algorithm, and merged together, separated by a whitespace character. We finally get the whole paragraph transcription p_{pg}.

4.4.3 Experiments

This section is dedicated to the evaluation of the VAN for paragraph recognition. We show that the VAN reaches state-of-the-art results on each dataset. We study the need for pre-training on isolated text lines of the target dataset. We show that this can be avoided by using a pre-trained VAN on another dataset. It enables the model to be trained on the target dataset with the paragraph-level ground truth only, without the need for line segmentation ground truth, which is a considerable practical advantage. We study the different stopping strategies on the IAM dataset. We also provide a visualization of the attention process.

4.4.3.1 Comparison with state-of-the-art paragraph-level approaches

The results presented in this section are given for the VAN, with pre-training on line images and using the learned-stop strategy. The following comparisons are made with approaches under similar conditions, *i.e.* without the use of external data (to model the language for example) and at paragraph level.

Comparative results with state-of-the-art approaches on the RIMES 2011 dataset are given in Table 4.4. The VAN achieves better results on the test set compared to the other approaches with a CER of 1.91% and a WER of 6.72%.

Table 4.4: Recognition results of the VAN and comparison with paragraph-level state-of-the-art approaches on the RIMES 2011 dataset.

Architecture	CER (%) valid	WER (%) valid	CER (%) test	WER (%) test	# Param.
[82] CNN+MDLSTM[a]	2.5	12.0	2.9	12.6	
[107] RPN+CNN+BLSTM+LM			2.1	9.3	
[145] SPAN (FCN)	3.56	14.29	4.17	15.61	19.2 M
[146] VAN (FCN+LSTM)[a]	1.83	6.26	**1.91**	**6.72**	2.7 M

[a] With line-level attention.

Table 4.5 shows the results compared to the state of the art on the IAM dataset. As one can see, once again the VAN also achieves new state-of-the-art results with a CER of 4.45% and a WER of 14.55% on the test set. One can notice that we use more than 6 times fewer parameters than in [144] and [145] with 2.7 M compared to 16.4 M and 19.2 M.

Table 4.5: Comparison of the VAN with the state-of-the-art approaches at paragraph level on the IAM dataset.

Architecture	CER (%) valid	WER (%) valid	CER (%) test	WER (%) test	# Param.
[83] CNN+MDLSTM[a]			16.2		
[82] CNN+MDLSTM[b]	4.9	17.1	7.9	24.6	
[102] RPN+CNN+BLSTM[c]	13.8		15.6		
[104] RPN+CNN+BLSTM			8.5		
[107] RPN+CNN+BLSTM+LM			6.4	23.2	
[144] GFCN			4.7		16.4 M
[145] SPAN (FCN)	3.57	15.31	5.45	19.83	19.2 M
[146] VAN (FCN+LSTM)[a]	3.02	10.34	**4.45**	**14.55**	2.7 M

[a] With character-level attention.
[b] With line-level attention.
[c] Results are given for page level.

To our knowledge, except for the SPAN, there are no results reported in the literature on the READ 2016 dataset at paragraph or page level. The recognition results are presented in Table 4.6. The VAN reaches better results than the SPAN with a CER of 3.59% and a WER of 13.94%.

Table 4.6: Comparison of the VAN with the state-of-the-art approach for the READ 2016 dataset at paragraph level.

Architecture	CER (%) validation	WER (%) validation	CER (%) test	WER (%) test	# Param
[145] SPAN (FCN)	5.09	23.69	6.20	25.69	19.2 M
[146] VAN (FCN+LSTM)	3.71	15.47	**3.59**	**13.94**	2.7 M

The proposed Vertical Attention Network achieves new state-of-the-art results on these three different datasets. For fair comparison with the other competitive approaches of the literature we highlight some comparative features in Table 4.7. In this table, the approaches are analyzed with regards to the following features: 1- use of an explicit text region segmentation process 2- minimum segmentation level used (whether it is for pre-training, data augmentation or training itself) 3- number of hyperparameters adapted from one dataset to another (except for the last decision layer which is dependent of the alphabet size) 4- use of curriculum learning 5- use of data augmentation. Curriculum learning, which consists in progressively increasing the number of lines in the input images, can be considered as data augmentation (crop technique). It is important to note that the use of line segmentation ground truth during training is costly due to the human effort involved to create them; this is even more costly for word segmentation. Data augmentation, for its part, does not require any human effort.

Table 4.7: Training details of state-of-the-art approaches.

Approach	Explicit segmentation	Min. segment. label used	Hyperparam. adaptation	Curriculum learning	Data augmentation
[102] RPN+CNN+BLSTM	✓	Word	✗	✗	✗
[104] RPN+CNN+BLSTM	✓	Word	✗	✗	✓
[107] RPN+CNN+BLSTM	✓	Line	✗	✗	✓
[144] GFCN	✗	Paragraph	6	✗	?
[82] CNN+MDLSTM	✗	Line	✗	✓	✓
[83] CNN+MDLSTM	✗	Line	✗	✓	✓
[145] SPAN (FCN)	✗	Line	✗	✗	✓
[146] VAN (FCN+LSTM)	✗	Line	✗	✗	✓

The approach proposed in [144] is the only one that does not use any segmentation label neither at line nor at word level. However, it is also the only one that requires the architecture to be adapted to each dataset. More specifically, the height and width of the input and of two intermediate upsampling layers are tuned for each dataset. It means that those 6 hyperparameters must be tuned manually to reach the performance reported, which makes the architecture not generic at all. On the contrary, the VAN architecture remains the same for every dataset in all the experiments. Another important point is that the use of line-level segmentation ground truth is not inherent to our approach, as it was only introduced as a pre-training step. As we will see in the following section, pre-training the model at line level is not mandatory. Indeed, similar performance can be obtained with paragraph-level cross-dataset pre-training, without the need for line segmentation ground truth from the target dataset.

4.4.3.2 Impact of pre-training

In this section, we study the impact of pre-training on the VAN, using the learned-stop approach. The dataset used for pre-training is named source dataset, and the dataset on which we want to evaluate the model is named target dataset. We conducted three kinds of experiments. They are intended to evaluate the requirement of line-level segmentation labels for the target dataset to reach the best performance.

A first training strategy consists in training the architecture at paragraph level directly on the target dataset, from scratch. A second strategy consists in training the architecture at paragraph level on the target dataset, with pre-trained weights for the encoder and the convolutional layer of the decoder, as detailed in Section 4.4.2. In this case, pre-training is carried out on the isolated text line images of the target dataset, prior to train the VAN at paragraph level. Here, the source dataset and the target dataset are the same. This pre-training approach is referred to as line-level pre-training. A third training strategy consists in training the architecture at paragraph level on the target dataset with weights that are initialized with those of another VAN. This other VAN is trained on a source dataset, with the second strategy. The idea is not to use any segmentation label from the target dataset. In this case the training strategy is referred to as cross-dataset pre-training.

Figure 4.13 shows the evolution of the CTC loss on the IAM dataset during training from scratch and training with line-level pre-training. We also highlight the impact of dropout when training from scratch. The recognition performance is given on the test set in Table 4.8. As was expected, we can clearly notice that training the model from scratch is feasible,

but it takes much time to converge and it comes at the cost of an important increase of the CER, from 4.45% up to 7.06%. When training from scratch, the use of dropout highly slows down the convergence but it leads to a lower CER of 7.06% compared to 8.06%. This experiment shows us that state-of-the-art results are reached with line-level pre-training.

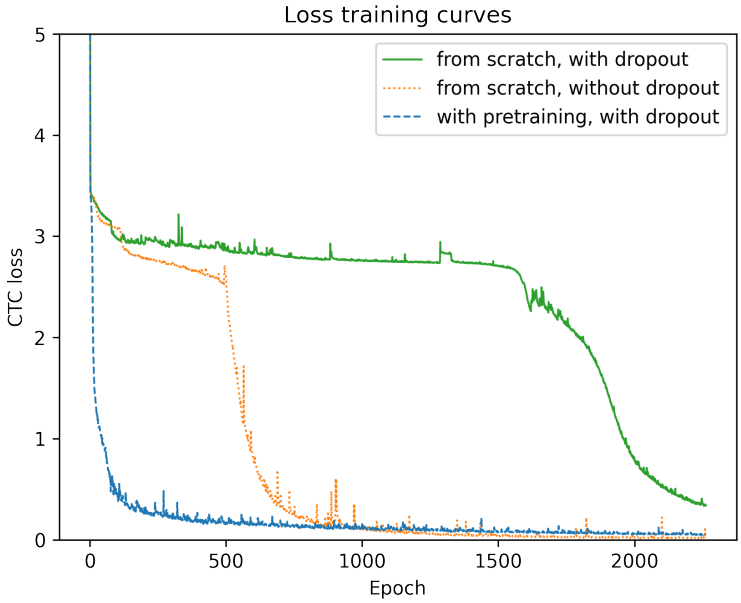


Figure 4.13: CTC training loss curves comparison for the VAN, with and without pre-training, on the IAM dataset.

Table 4.8: Impact of the pre-training on lines for the VAN. Results are given on the test set of the IAM dataset.

pre-training	Dropout	CER (%)	WER (%)	Training time
✗	✗	8.06	25.38	1.39 d
✗	✓	7.06	22.65	1.77 d
✓	✓	**4.45**	**14.55**	0.63 d

Table 4.9 reports the recognition performance of the third training strategy, using cross-dataset pre-training. In this table we report the performance of the VAN on the three datasets when pre-trained on one of the two other datasets. One can notice that this pre-training strategy performs almost similarly as line-level pre-training for every dataset, but without using any segmentation label of the target dataset. This is especially true with the two similar datasets RIMES 2011 and IAM, for which the two pre-training strategies reach very similar CER: 1.97% compared to 1.91% for RIMES 2011 and 4.55% compared to 4.45% for IAM. Although cross-dataset pre-training is a bit less efficient on READ 2016, it still leads to competitive results. This may be explained by the differences between this dataset (RGB color encoding, historical manuscripts and language) and the other datasets (RIMES 2011 or IAM) on which pre-training is performed.

Table 4.9: Comparison between cross-dataset pre-training and line-level pre-training for the VAN. Results are given on the test sets.

Source dataset	RIMES 2011 CER (%)	IAM CER (%)	READ 2016 CER (%)
Cross-dataset pre-training			
RIMES 2011	✗	4.55	4.08
IAM	1.97	✗	4.14
READ 2016	2.36	5.20	✗
Line-level pre-training			
Target dataset	**1.91**	**4.45**	**3.59**

These experiments demonstrate the importance of pre-training for the VAN. However, we show that we can alleviate the need for line-level segmentation label of the target dataset through cross-dataset pre-training. We assume that the important pre-training effect is mainly due to the attention mechanism. Indeed, the authors of [82, 83], who also proposed attention-based models, used curriculum learning to tackle convergence issues. We assume this is due to the direct relation between recognition and implicit segmentation (through soft attention). When the encoder is pre-trained, the VAN only has to learn the attention module and the decoder, as the pre-trained features may contain all the information needed for the recognition of the characters. Considering random distribution of the attention weights over the vertical axis at first, training will progressively increases values of weights where the features correspond to the correct characters. But when the encoder is randomly initialized, the features extraction must also be learned, which slows the whole training. The use of dropout makes this phenomenon even worse, but it is essential in this architecture to avoid overfitting. It seems that cross-dataset pre-training skips this issue since the attention mechanism is already learned and the encoder only needs to be fine tuned.

Finally, we can notice that without introducing any pre-training, the VAN architecture is able to converge using the paragraph annotations only, but it is not competitive anymore compared to the state of the art. As a preliminary conclusion, we can highlight the capacity of the VAN to achieve state-of-the-art performance without no need to adapt the architecture to each dataset considered. Moreover, cross-dataset pre-training allows to reach similar results compared to line-level pre-training, without the need to use any line-level segmentation ground truth from the target dataset.

4.4.3.3 Learning when to stop

We now compare the three stopping strategies mentioned in Section 4.4.1.2, namely fixed-stop, early-stop and learned-stop approaches.

Performance evaluation of these three methods is given in Table 4.10 for comparison purpose. Line-level pre-training is used for each approach, as detailed in Section 4.4.2. We define d_{mean}, as the average of the absolute values of the differences between the actual number of lines in the image n_{i} and the number of recognized lines n_{r}. This metric is used to evaluate the efficiency of the early-stop and learned-stop approaches. For K images in the dataset:

$$d_{\text{mean}} = \frac{1}{K} \sum_{k=1}^{K} |n_{\text{i}_k} - n_{\text{r}_k}|. \tag{4.11}$$

As one can see, equivalent CER and WER are obtained for each stopping strategy. Moreover, the prediction time is not significantly impacted since data formatting, tensor initialization and encoder-related computations take much longer than the recurrent process, which is made up of only a few layers at each iteration.

Table 4.10: Comparison between fixed-stop, early-stop and learned-stop approaches with the VAN on the test set of the IAM dataset.

Stop method	CER (%)	WER (%)	Train. time	Pred. time	d_{mean}
Fixed	**4.41**	14.69	1.20 d	33 ms	20.32
Early	**4.41**	**14.39**	0.41 d	32 ms	0.02
Learned	4.45	14.55	0.63 d	32 ms	0.03

Figure 4.14 illustrates the evolution of the CTC loss for the three approaches. One should keep in mind that the comparison is biased since the approaches do not iterate the same number of times for a same example. However, one can clearly notice that the early-stop and learned-stop approaches converge similarly, in contrast to the fixed-stop approach, which requires far more epochs to converge. But in the end, they reach almost identical CER.

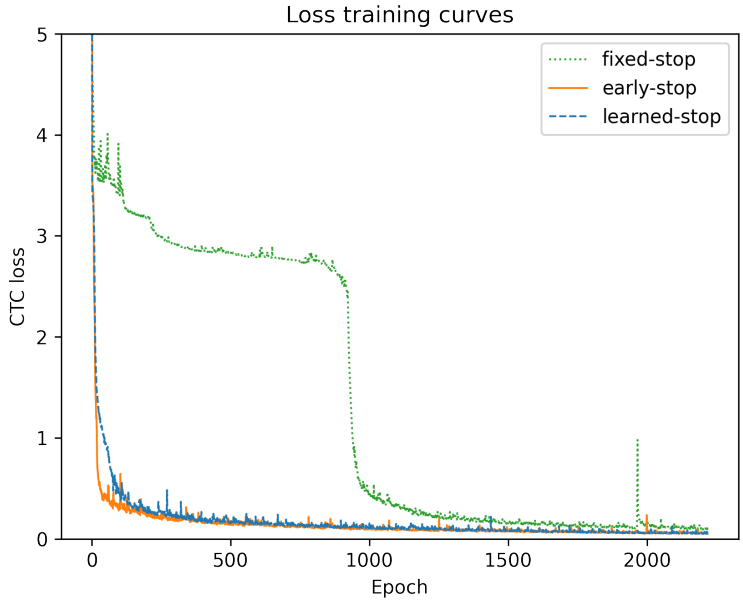


Figure 4.14: CTC training loss curves comparison for the VAN for each stopping approach on the IAM dataset.

Figure 4.15 compares the d_{mean} for the early-stop and learned-stop approaches on the IAM validation dataset. For visibility, the fixed-stop approach curve, which is a plateau at

$d_{\text{mean}} = 20.32$, is not depicted in this figure. The learned-stop approach leads to a faster convergence of d_{mean} compared to the early-stop approach, whose curve is less stable. It results in a d_{mean} of 0.03 for the learned-stop approach and 0.02 for the early-stop approach for the test set of IAM.

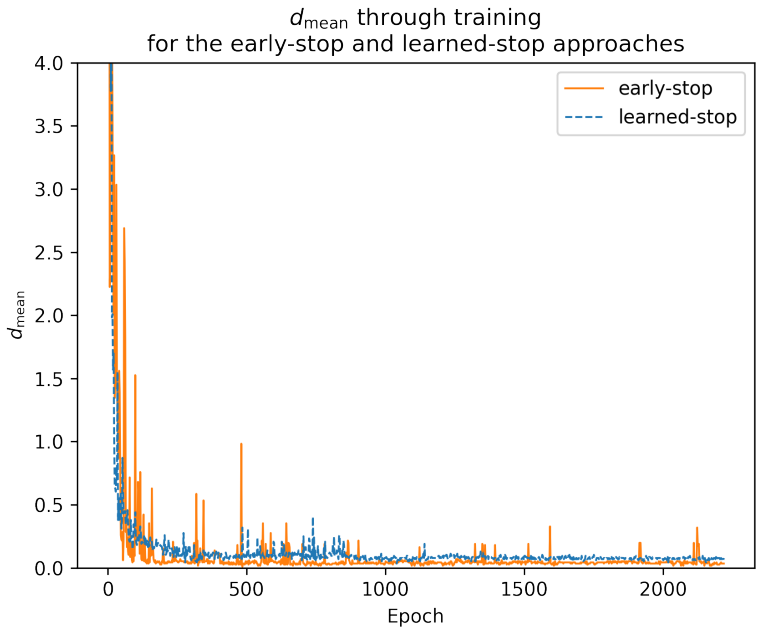


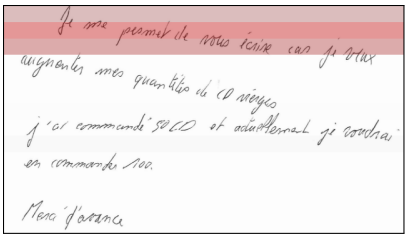
Figure 4.15: Comparison of the evolution of d_{mean} (the mean difference between the true and estimated number of lines in the image) on the validation set of IAM dataset for the early-stop and learned-stop approaches.

With the learned-stop approach, the model successfully learns both tasks: it recognizes text lines with a state-of-the-art CER of 4.45% and it determines when to stop with a high precision since the d_{mean} on the test set is only 0.03. It means that, on average, one line is missed or processed twice every 33 paragraph images. We choose to keep this stopping strategy since it slightly improves the stability of the convergence through the epochs while achieving nearly identical results.

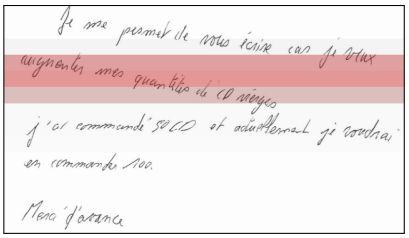
4.4.4 Visualization of the vertical attention

Figure 4.16 shows the processing steps of an image from the RIMES 2011 validation set with a complex layout. Images represent the attention weights of the 5 attention iterations, each one recognizing a line. The intensity of the weights is encoded with the transparency of the red color. Given that attention weights are only computed along the vertical axis, the intensity is the same for every pixel at the same vertical position. Attention weights are rescaled to fit the original image height; indeed, attention weights are originally computed for the features height, which is 32 times smaller. The recognized text lines are given for each iteration, below the image.

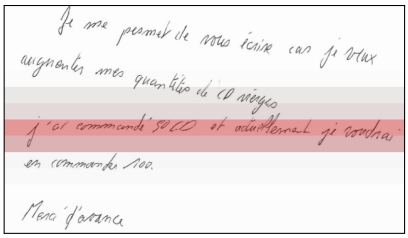
As one can see, the VAN has learned the reading order, from top to bottom. The



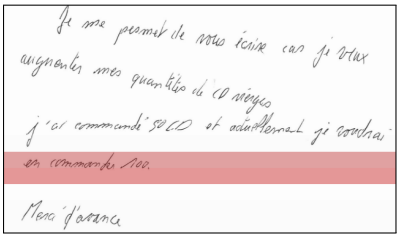
1: Je me permet de vous écrire cas je vMux



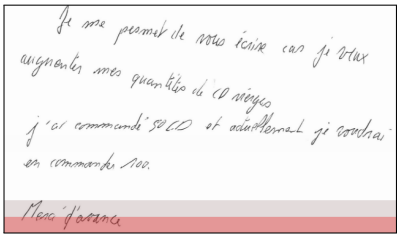
2: augmentes mes quantités de CD vierges



3: j'ai commandé 50 CD et aduellement je voudrai



4: en commander 100.



5: Merci d'avance

Figure 4.16: Attention weights visualization on a sample of the RIMES 2011 validation set. Recognized text is given for each line and errors are shown in bold.

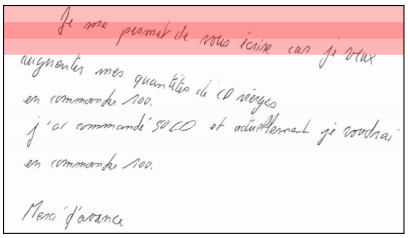
attention weights clearly focus on text lines following this reading order. Attention weights focus mainly on one features line, with smaller weights for the adjacent vertical positions.

One can notice that, sometimes, the focus is not perfectly centered on the text line. This may be due to rescaling, but this can also be a normal behavior due to the large size of the receptive field, which enables to manage slightly inclined lines. The second iteration shows this phenomenon very well with only one misrecognized character on an inclined line. Furthermore, one can notice that the attention is less sharp when the layout is more complex between two successive lines, as in the third image, but it does not disturb the recognition process however.

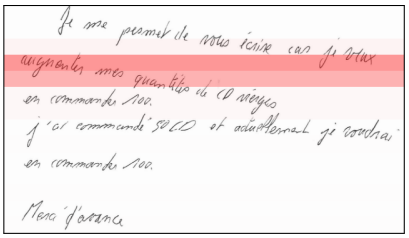
One should note that processing inclined text lines is only possible when the lines, although inclined, do not share the same vertical position. The VAN cannot handle the case where lines would overlap vertically, because the attention weights would mix the two lines in this case. This is illustrated on Figure 4.17 with a fake input image, which consists in the previous image in which we duplicated a text line.

As one can see, some text is predicted twice ("écrire car je veux") and other is ignored ("de CD vierges"). But one can note that only the text part which includes overlaps in horizontal projection is altered: it does not affect the quality of the recognition for the remaining text.

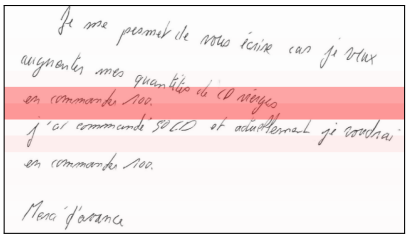
So far, we have provided strong results in favor of the Vertical Attention Network by achieving state-of-the-art performance on three datasets. We also evaluated the need for pre-training and the efficiency of the stopping strategies we propose.



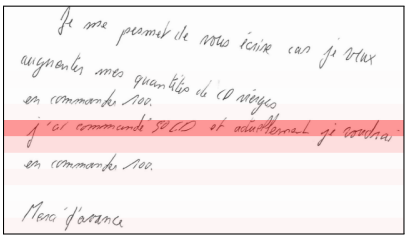
1: Je me permet de vous écrire car je v**M**ux



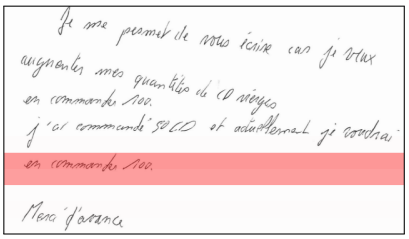
2: augmente**s** mes quantités **d'is vièrire cas je MuX**



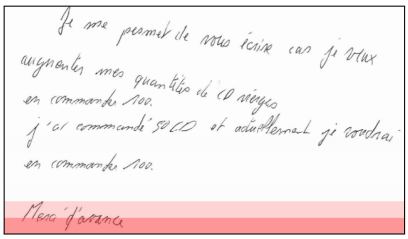
3: en commander 100.



4: J'ai commandé 50 CD et a**d**uellement je voudrai



5: en commander 100.



6: Merci d'avance

Figure 4.17: Attention weights visualization on a synthetic sample of the RIMES 2011 validation set. Recognized text is given for each line and errors are shown in bold.

4.4.5 Additional experimental studies

We now provide additional results which show the superiority of the VAN on many criteria compared with a standard line-level three-step approach (line segmentation followed by ordering and recognition). We also highlight the positive contribution of the VAN applied to paragraph images compared with the single-line recognition approaches. Finally, we highlight the positive effect of the proposed dropout strategy (Diffused Mix Dropout) compared to standard ones.

4.4.5.1 Comparison with the standard line-level three-step approach

In this section, we compare the VAN with the standard line-level three-step approach on the IAM dataset. In this respect, two different models are used for the line-level segmentation and recognition stages; the ordering stage is performed through a rule-based algorithm. Both models are trained separately and they do not use any pre-training strategy since they are already working at line level.

The line segmentation model, depicted in Figure 4.18 follows a U-net shape architecture and is based on the VAN FCN encoder. Indeed, the features $\boldsymbol{f}$ are successively upsampled to match the the shape of the feature maps of CB_5, CB_4, CB_3, CB_2 and CB_1 (Figure 4.7). This upsampling process is handled by Upsampling Blocks (UB). UB consists in Depthwise Separable Convolution (DSC) layer followed by DSC Transpose layer and instance normalization. This block also includes DMD layers. Each UB output is concatenated with the feature maps from its corresponding CB. A final convolutional layer outputs only two feature maps, to classify each pixel of the original image between text and background. The

line-level HTR model corresponds to the one used for the VAN pre-training (Figure 4.9).

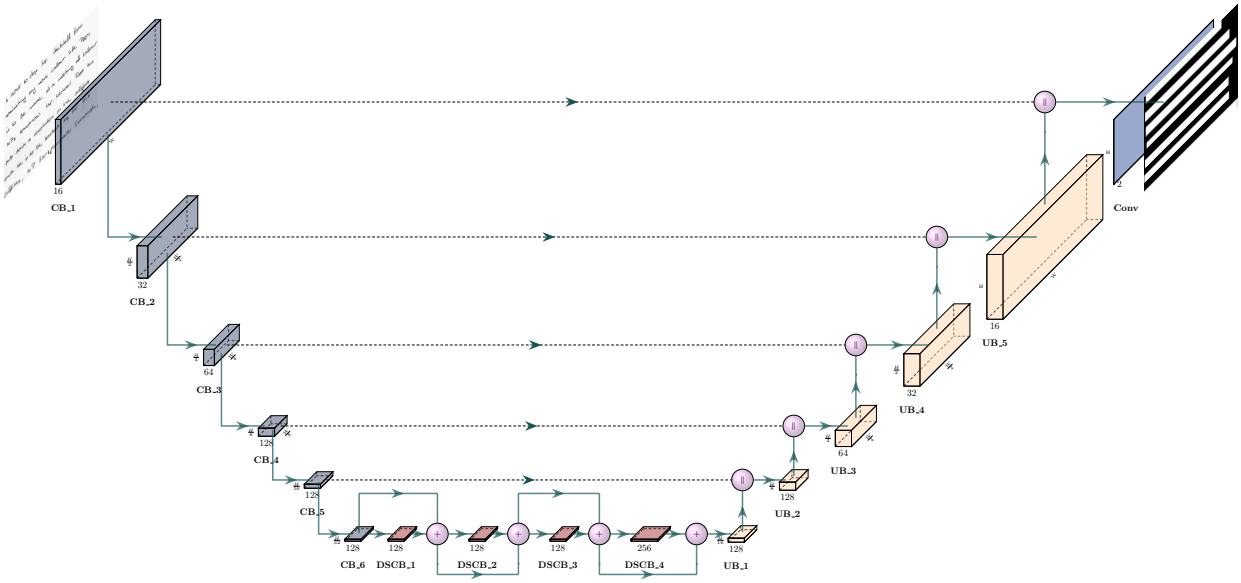


Figure 4.18: U-net architecture for text line segmentation.

The line segmentation model is trained on the paragraph images at pixel level with ground truth of line bounding boxes. It is trained with the cross-entropy loss. The line-level HTR model is trained on the isolated text line images with the CTC loss. We used a mini-batch size of 8 for the segmentation task and of 16 for the HTR.

We now detail the three steps of this approach. In a first step, paragraphs are segmented into lines:

- Ground truth bounding boxes are modified in order to avoid overlaps: we divide their height by 2.
- A paragraph image is given as input of the network.
- A 2-class pixel segmentation (text or background) is output from the network.
- Adjacent text-labeled pixels are grouped to form connected components.
- Bounding boxes are created as the smallest rectangles containing each connected component; their height is multiplied by 2.
- The input image is cropped using those bounding boxes to generate line images.

In a second step, the line images are ordered given their vertical position, from top to bottom. The last stage consists in recognizing the line images with the line-level HTR model, trained on the IAM dataset. The recognized text lines are concatenated to compute the CER and the WER at paragraph level.

The performances of both tasks (segmentation and recognition) taken separately and together are shown in Table 4.11. As one can see, the results are good for both tasks

separately: we get 81.51% for the IoU and 85.09% for the mAP concerning the text line segmentation task; and a CER of 5.01% and a WER of 16.49% for the line-level HTR. However, when we take the output of the segmentation as input for the HTR model, it leads to a CER increase of 1.54 points. Indeed, the errors of the segmentation stage induce recognition errors.

Table 4.11: Results of the three-step approach on the test set of IAM.

Architecture	IoU (%)	mAP (%)	CER (%)	WER (%)	# Param.
Line seg. model	81.51	85.09	✗	✗	1.8 M
Optical Character Recognition (OCR) on lines	✗	✗	5.01	16.49	1.7 M
Three-step approach	81.51	85.09	6.55	18.54	1.8+1.7M

We can now compare the VAN to the three-step approach. Comparison on the IAM test set is summarized in Table 4.12. First, one can notice that the VAN reaches a better CER of 4.45% compared to 6.55%. Prediction time is computed as the average prediction time, on the test set, to process a paragraph image. As one can see, even though the segmentation step is without recurrence, it requires much more time for prediction due to the formatting of the input required for the HTR, including the bounding boxes extraction from the original images. Moreover, it cumulates prediction times of the two models involved. Despite its recurrent process, the total prediction time for the VAN is shorter than that of the three-step approach since one iteration is very fast. In addition, it implies fewer parameters, this is notably due to the two models required by the three-step approach. Except for the training time, which is a bit higher for the VAN, it only provides advantages compared to the three-step approach.

Table 4.12: Comparison of the three-step approach with the VAN, results are given for the test set of the IAM dataset.

Architecture	CER (%)	WER (%)	# Param.	Training time	Prediction time
Three-step	6.55	18.54	1.8+1.7 M	0.03+0.59 d	749+28 ms
VAN	**4.45**	**14.55**	2.7 M	0.59+0.63 d	32 ms

4.4.5.2 Line-level analysis

In this section, we compare the results of the VAN with the state-of-the-art approaches evaluated in similar conditions *i.e.* at line level and without using external data. We also provide results for the line-level HTR model (Figure 4.9), which has the same encoder, and for the VAN applied at paragraph level. The aim is to highlight the contribution of the VAN processing full paragraph images instead of isolated lines.

To have a fair comparison between the results obtained at paragraph and line levels, one should consider the difference in the ground truth: the paragraph transcriptions contain line breaks (or space characters in our case) between the different line transcriptions. Table 4.13 shows the VAN results difference at paragraph level when removing the interline characters from the metrics. As one can see, the removal of the interline character leads to an increase

of the CER of 0.24, 0.09 and 0.24 on the test set of RIMES 2011, IAM and READ 2016 respectively. The WER is not impacted by this modification. In the following tables of this section, we will use the results without considering interline characters for fair comparison with line-level approaches.

Table 4.13: VAN results at paragraph level with and without interline characters in ground truth for RIMES 2011, IAM and READ 2016 dataset.

Dataset	Line break	CER (%) valid	WER (%) valid	CER (%) test	WER (%) test
RIMES 2011	✓	**1.83**	6.26	**1.91**	6.72
	✗	2.10		2.15	
IAM	✓	**3.02**	10.34	**4.45**	14.55
	✗	3.07		4.54	
READ 2016	✓	**3.71**	15.47	**3.59**	13.94
	✗	4.01		3.83	

Table 4.14 shows state-of-the-art results on the RIMES 2011 dataset at line level. We report competitive results with a CER of 3.04% for the line-level HTR model and 3.08% for the VAN on the test set compared to the model of [124] which reached 2.3%. On should notice that Puigcerver *et al.* [124] does not use exactly the same dataset split for training and validation. In conclusion we can highlight the performance of the VAN obtained at paragraph level which achieves a CER of 2.15% on the test set, which corresponds to decreasing the CER by 0.93 compared to processing isolated lines.

Table 4.14: Comparison of the VAN with the state of the art on the line-level RIMES 2011 dataset.

Architecture	CER (%) valid	WER (%) valid	CER (%) test	WER (%) test	# Param.
[122] MDLSTM+LM			2.8	**9.6**	
[124] CNN+BLSTM[a]	2.2	9.6	**2.3**	**9.6**	9.6 M
[146] Line-level HTR model (FCN)	2.20	6.26	3.04	8.32	1.7 M
[146] VAN on lines (FCN+LSTM)	1.97	6.09	3.08	8.14	2.7 M
[146] VAN on paragraphs (FCN+LSTM)	2.10	6.26	**2.15**	**6.72**	2.7 M

[a] This work uses a slightly different split (10,203 for training, 1,130 for validation and 778 for test).

Comparison with state-of-the-art results on the IAM dataset is presented in Table 4.15. We reach competitive results with a CER of 4.97% on the test set. Models proposed in [129] and [131] reach similar results but the former implies a large number of parameters compared to the VAN and the latter is more complex including a recurrent process with attention at character level. It should be noticed however that, in [124, 131, 129], the authors use a slightly different split from ours. As a matter of fact, since the VAN training implies pre-training at line level, it was not possible to use the same split since some lines for training and validation are extracted from the same paragraph image for example. On the IAM dataset, the VAN also reaches a better CER at paragraph level than at line level with 4.54% compared to 4.97%.

The results on the READ 2016 dataset are gathered in Table 4.16. We reached state-of-the-art CER on the test set with 4.10% compared to 4.66% for [131]. Again, the paragraph-

Table 4.15: Comparison of the VAN with the state of the art at the line level on the IAM dataset.

Architecture	CER (%) validation	WER (%) validation	CER (%) test	WER (%) test	# Param.
[122] CNN+MDLSTM+LM	**2.4**	**7.1**	**3.5**	**9.3**	2.6 M
[124] CNN+BLSTM[a]	3.8	13.5	5.8	18.4	9.3 M
[129] GFCN[a]	3.3		4.9		$>$ 10 M
[131] CNN+BLSTM[a]			4.87		
[146] Line-level HTR model (FCN)	3.37	11.52	5.01	16.49	1.7 M
[146] VAN on lines (FCN+LSTM)	3.15	10.77	4.97	16.31	2.7 M
[146] VAN on paragraphs (FCN+LSTM)	3.07	10.34	4.54	14.55	2.7 M

[a] These works use a slightly different split (6,161 for training, 966 for validation and 2,915 for test).

level VAN reaches better results than the VAN applied at line level with a CER of 3.83%.

Table 4.16: Comparison of the VAN with the state-of-the-art line-level recognizers on READ 2016 dataset.

Architecture	CER (%) validation	WER (%) validation	CER (%) test	WER (%) test	# Param.
[131] CNN+BLSTM			4.66		
[3][a] CNN+MDLSTM+LM			4.8	20.9	
[3][b] CNN+RNN			5.1	21.1	
[146] Line-level HTR model (FCN)	4.49	18.22	4.25	17.14	1.7 M
[146] VAN on lines (FCN+LSTM)	4.42	18.17	**4.10**	**16.29**	2.7 M
[146] VAN on paragraphs (FCN+LSTM)	4.01	15.47	**3.83**	**13.94**	2.7 M

[a] results from RWTH.
[b] results from BYU.

In conclusion, one can notice that the VAN, applied on isolated lines, performs at least similarly as the line-level model, for each dataset (except for RIMES with a small CER increase of 0.04 points). It also achieves state-of-the art results at line level on the READ dataset.

The results also highlight the superiority of the Vertical Attention Network on the RIMES 2011, IAM and READ 2016 datasets, applied to whole paragraph images, compared to isolated lines. Multiple factors can explain this result: segmentation ground truth annotations of text lines are prone to variations from one annotator to another, bringing variability that is not present when dealing with paragraph images directly. Indeed, the model implicitly learns to segment the lines so it does not have to adapt to pre-formatted lines; it uses more context (with a large receptive field) and uses it to focus on the useful information for the recognition purpose. Moreover, the VAN decoder contains a LSTM layer that may have a positive impact acting as a language model without any loss of context when moving from one line to the next, when producing the output character sequence.

A key element to reach such results with a deep network is to use efficient regularization strategies. We discuss the new dropout strategy we propose in the following paragraph.

4.4.5.3 Dropout strategy

We use dropout to regulate the network training and thus avoid overfitting. We defined Diffused Mix Dropout (DMD) in Section 4.3.1.1 to improve the results of the model. We carried out some experiments to highlight the contribution of DMD over commonly used standard and 2d dropout layers. Experiments are performed with the line-level HTR model and the VAN on the IAM dataset; VAN is pre-trained with weights from the corresponding line-level model. Results for the test set are shown in Table 4.17. The columns from left to right correspond respectively to the number of dropout layers per block (CB and DSCB), the type of dropout layer used (mix, standard or spatial), the associated dropout probabilities, the use of the diffuse option (using only one or all dropout layers per block) and the CER and WER for both models.

Table 4.17: Dropout strategy analysis. Results are given for the IAM test set.

	#	type	p	diffused	line-level model		VAN	
					CER (%)	WER (%)	CER (%)	WER (%)
Baseline	3	mix	0.5/0.25	✓	**5.01**	**16.49**	4.45	**14.55**
(1)	3	std.	0.5	✓	5.24	17.23	4.46	14.88
(2)	3	2d	0.25	✓	5.38	17.64	4.72	15.63
(3)	1	mix	0.5/0.25	✗	5.33	17.70	**4.43**	15.25
(4)	1	std.	0.5	✗	5.56	18.40	4.78	15.91
(5)	1	2d	0.25	✗	5.70	18.92	4.93	16.80
(6)	3	mix	0.5/0.25	✗	6.76	21.13	6.32	19.73
(7)	3	mix	0.16/0.08	✗	6.71	20.91	4.64	15.36
(8)	3	mix	0.16/0.08	✓	7.51	23.60	5.57	18.50

In (1) and (2), Mix Dropout layers are respectively replaced by standard and 2d dropout layers, preserving their corresponding dropout probability. Using Mix Dropout leads to an improvement of 0.23 points of CER compared to standard dropout and of 0.37 compared to 2d dropout for the line-level model. These improvements are lower for the VAN with 0.01 and 0.27 points.

In (3), only one Mix Dropout is used, after the first convolution of the blocks, leading to a higher CER than the baseline, with a difference of 0.34 points for the line-level model. The CER is decreased by 0.02 points for the VAN but the CER is increased by 0.70 points. In (4) and (5), we are in the same configuration as (3) *i.e.* with only one dropout layer per block. Mix Dropout is superseded by standard dropout in (4) and by 2d dropout in (5) resulting in an increase of the CER compared to (3). This shows the positive impact of Mix Dropout layers in another configuration.

In (6) and (7), Mix Dropout layers are set at each of the three positions *i.e.* they are all used at each execution, contrary to the baseline, which uses only one dropout layer per execution. While (6) keeps the same dropout probabilities, (7) divides them by 3. In both cases, the associated CER are higher than the baseline.

Finally, in (8), we are in the same context than the baseline, but dropout probabilities are divided by 3, leading to higher CER.

We can conclude that our dropout strategy leads to a CER improvement of 0.55 points for the line-level model and of 0.33 for the VAN, when compared to (4) and (5) that do not use Mix Dropout or diffuse option.

4.4.6 Discussion

We proposed the Vertical Attention Network as a novel end-to-end encoder-decoder segmentation-free architecture using hybrid attention for the task of handwritten paragraph recognition. As we have seen, the VAN achieves state-of-the-art results on multiple datasets at paragraph level. However, there is one point that should be notice. Modern deep neural systems involve many training strategies (hyperparameters, optimizer, regularization strategies, preprocessings, data augmentation techniques, transfer learning, curriculum learning, and many others). This makes the comparison between architectures very difficult as some training tricks are more suited for some architectures than some others. This is why one should be convinced that the state-of-the-art results obtained by the VAN are due to the whole proposition, including training strategies, and not only to the VAN architecture. However, we have provided experimental results that show the interest of the proposed training strategies of the generic VAN architecture.

We compared different stopping strategies and showed that the VAN can learn to detect the end of paragraph. This additional task has no significant impact on the performance and slightly improves the stability through training. We also compared favorably the VAN to a standard two-step approach and showed the positive impact of processing paragraph-level images compared with line-level ones, for this architecture. The new dropout strategy we propose enabled to reach even better results.

Moreover, the VAN has multiple advantages. The VAN is robust: whether it is at paragraph or line level, and no matter the dataset used, we did not adjust any hyperparameter for each dataset. The VAN takes input of variable sizes, so it could handle whole page images without any modification. As mentioned previously, the VAN can handle slightly inclined lines. However, it is limited to layouts in which there is no overlap between lines on their horizontal projection. Indeed, this case remains to be solved. A standard n-gram language model could process the outputs of the VAN architecture but its impact on the performance remains to be determined through experiments.

However, there is still room for improvement. Notably, we showed that the VAN needs pre-training on isolated text lines of the target dataset to reach state-of-the-art results. But the need for line-level annotations is not inherent to the VAN, this is only related to this pre-training step. As a matter of fact, we demonstrate that pre-training on another dataset (cross-dataset pre-training) can alleviate this issue, even if the datasets are really different. Indeed, for the three datasets, cross-dataset pre-training leads to results similar to those from line-level pre-training, but without using any line-level annotation from the target dataset.

The VAN should be considered to process single-column text document only. As a matter of fact, as it is the case for [82] and supposedly [144], the models are designed and limited to process single-column multi-line text documents with relatively horizontal text lines.

4.5 Conclusion

We have presented two different architectures, the SPAN and the VAN, which follow two totally different approaches to deal with HTR at paragraph-level. Although they reach competitive and state-of-the-art results on 3 public datasets for the SPAN and the VAN respectively, both approaches present the same limitations:

- They both rely on a pre-training stage based on the use of line-level segmentation annotations: this reduces the interest of dealing with paragraph-level images. The only end-to-end model of the literature which does not use any line-level segmentation annotation during training ([144]) compensate with the tuning of dataset-specific hyperparameters, making it not generic at all.
- They are, by design, limited to single-column text. It means that, although it has not been tested, they could theoretically handle whole documents with simple layout such as single-column pages. However, they cannot handle multi-column pages.
- These models are quite limited to tackle slanted text lines.

These issues could be partly solved with the use of a character-level attention mechanism. Indeed, with attention at character level, we do not use the *a priori* assumption that there can be only one text line per vertical position. In addition, the attention weight is not associated to a whole row of pixels or features: the attention mechanism can follow the slant of the text lines.

The authors of [72, 147] recently proposed a CNN combined with a transformer decoder module for HTR through a recurrent character-level attention process. However, the proposed approaches in the literature based on character-level attention are hardly competitive with a CER of 16.2% in [83] and 6.7% in [72] on the test set of the IAM dataset at paragraph-level. In addition, they rely on synthetic samples based on word segmentation annotations.

In the next chapter, we propose to handle the task of HTR at page and double-page level with a character-level attention model, also recognizing the layout, without using any physical layout annotations for training.

Chapter 5

Handwritten document recognition

A handwritten document is a complex structure composed of handwritten text blocks structured according to a specific layout. It can also contain non textual items such as images and tables. In this thesis, we define Handwritten Document Recognition (HDR) as the joint recognition of both text and layout. In Chapter 3 and Chapter 4, we have studied the task of Handwritten Text Recognition (HTR) through the use of a three-step approach at line and paragraph levels. Although Document Layout Analysis (DLA) is not necessary, it can be carried out during the segmentation step, leading to a semantic segmentation task. In this case, DLA and HTR are two tasks that are generally processed in an independent way in the literature. We assume that this does not seem relevant since these two tasks are closely related. In addition, this three-step approach suffers from multiple drawbacks. It relies on segmented entities which do not have a clear definition from the image point of view. For instance, text lines can be defined by X-heights, baselines, bounding boxes or polygons. This approach also requires segmentation annotations, which are very costly to produce. The resulting predictions accumulate errors between each stage, leading to high error rates. In this chapter, we propose to tackle the task of HDR by using a single end-to-end model, free from the aforementioned issues, and able to recognize both text and layout from a document image.

5.1 Problem statement

We defined HDR as the joint recognition of both text and layout. By this, we mean that the goal is not only to recognize all the text from the input document in a humanly correct order, but also to tag sub-sequences of the predicted sequence of characters given some layout classes. In other words, contrary to the three-step approach, whose segmentation (or DLA) stage consists in associating a layout class to some pixels of the input image, we propose to associate the layout class directly to a sequence of characters. This approach is depicted in Figure 5.1 with a letter example. As one can note, text parts, in addition to be recognized, are labeled with one of the following layout classes: sender coordinates, recipient coordinates, object, body and signature. In this example, layout classes correspond to logical layout entities (including some semantic) but it can also be physical layout entities such as header, footer, different title levels, section or paragraph. In this chapter, we only focus on

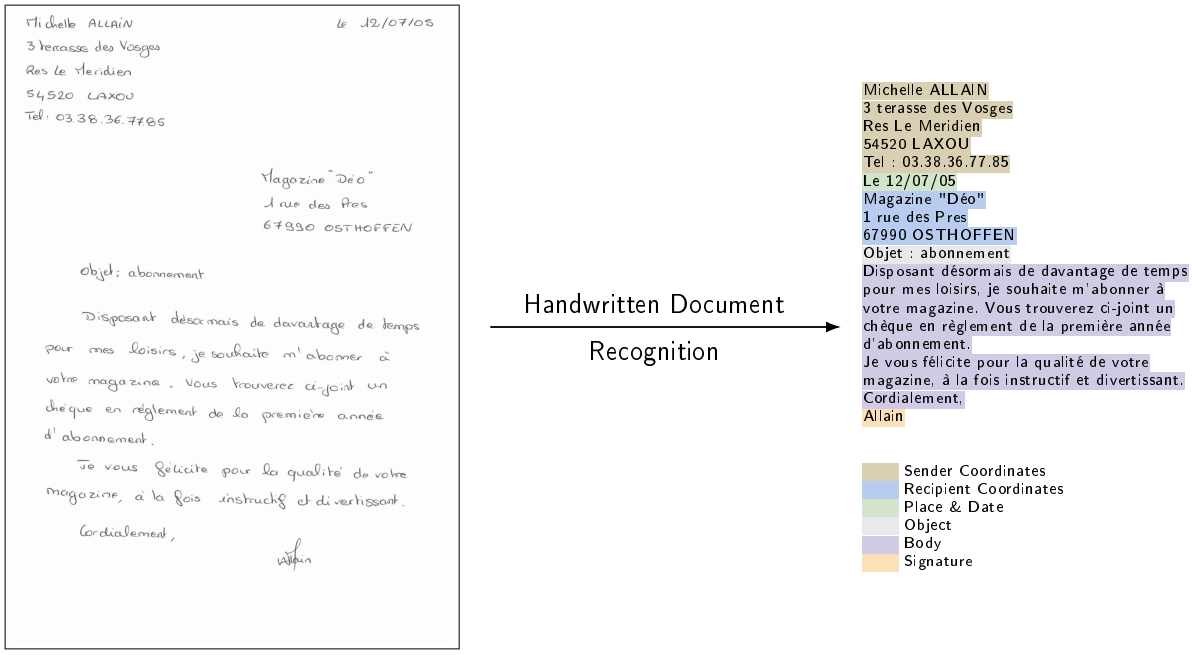


Figure 5.1: Overview of Handwritten Document Recognition.

textual entities.

Processing whole documents instead of isolated paragraphs raises new challenges:

- Paragraph-level models usually take benefits from the fact that all horizontal elements at the same vertical position belong to the same text line. This assumption is no longer valid at document-level.
- The input images are larger and the associated target sequences are longer for documents than for paragraphs, leading to more complex training procedures, especially in the case of attention-based models, which involves a growing need for GPU memory.
- While paragraphs are read in a monotonous order (characters are read from left to right and lines from top to bottom in case of common occidental languages for example), documents reading order is layout dependent *i. e.* paragraphs of a single-column document are read from top to bottom only, whereas a multi-column document is commonly read column by column, adding a horizontal constraint on the reading order.

5.2 Related works

Deep learning models are becoming more and more powerful and can now process entire documents. As we aim at recognizing both the text and the logical layout information of a handwritten document image, the task falls into both the HTR field and the DLA field, thus sharing links with document understanding. To our knowledge, we propose the first end-to-end approach for HDR.

Therefore, this section is dedicated to document understanding in a general way, and then focuses on layout analysis and handwriting recognition.

Document understanding

Document understanding includes a set of tasks whose purpose is to extract, classify, interpret, contextualize, and search information from documents. It implies, among others, DLA and Optical Character Recognition (OCR). But it also consists in understanding complex structures such as tables, schemes or images. This is a developing field and here are some related works. The authors of [148, 149] proposed Chargrid, a way to represent textual documents as a 2D representation of one-hot encoded characters, which are produced by an OCR. The idea is to keep the spatial information between the textual entities. Chargrid is then used as the input for a Key Information Extraction (KIE) task implying three sub-tasks: bounding box regression, semantic segmentation and box masking. In [150], the Chargrid paradigm is also used but the character's encoding are superseded by word embeddings through the use of the BERT model [12]. In [151], the authors tackle the task of Visually-rich Document Understanding (VDU). They used a transformer architecture applied on multiple modalities: OCR text and bounding boxes, and visual embedding. The authors of [152] proposed a transformer-based model able to tackle multiple tasks such as document classification, KIE and Visual Question-Answering (VQA). All these works imply the use of large datasets, mainly synthetic: they need a lot of information either as input or as ground truth annotation (notably for bounding boxes). One can notice that document understanding area is still in its early stages. It mainly focuses on 2D document representation and information extraction. Our work differs from document understanding in which we propose to extract all the text from a document; we do not aim at retrieving only specific information.

Document layout analysis

Related works for DLA were described in Section 4.2.1. DLA focuses on identifying physical regions of interest whether they are textual or not. It is driven by physical ground truth annotations accounting for semantic labels that are associated to document logical elements. HDR solely focuses on textual components for which we target their recognition and semantic labeling.

Handwritten text recognition

We presented related works for HTR in Section 3.2.3 (line-level) and Section 4.2.2 (paragraph-level). Among the paragraph-level HTR approaches, a single work has been devoted to the recognition of some layout items, in addition to the text. As a matter of fact, the model from [72] is trained to recognize, in addition to the text, the presence of non-textual areas: tables, drawings, math equations and deleted text. As one can note, this model recognizes non-textual items but it does not tag the predicted text with a layout class.

The main drawbacks of end-to-end HTR systems are as follows:

- The paragraph limitation. Except for [72], which uses a page-level private dataset, there is no work evaluating their model on a page-level dataset.
- The need for segmentation annotations. Even if the models are trained on paragraph-level images, line-level segmentation annotations are used to reach competitive results, either for pre-training ([145, 146]) or for synthetic data generation ([72, 147, 83]). The only work which does not use any line-level segmentation annotation [144] is limited to single-column text images and uses some dataset-specific hyperparameters to reach its competitive results.

We propose the Document Attention Network (DAN), an end-to-end transformer-based model for whole Handwritten Document Recognition, including the textual components and the logical layout information. This model is trained without using any segmentation label and we evaluate it on two public datasets at page and double-page levels: RIMES 2009 and READ 2016. To our knowledge, this is the first attempt that provides experimental results on such a task.

5.3 DAN: a Document Attention Network

We propose an end-to-end segmentation-free architecture for the task of HDR: the DAN. In addition to the text recognition, the model is trained to label text parts using begin and end tags in an XML-like fashion. This model is made up of a Fully Convolutional Network (FCN) encoder for feature extraction and a stack of transformer decoder layers for a recurrent token-by-token prediction process. It takes whole text documents as input and sequentially outputs characters, as well as logical layout tokens.

It is designed to handle whole documents with complex layouts such as double-column pages, and double-page documents, as depicted in Figure 5.2.

While the textual content can be represented as a sequence of characters, the layout is represented as an oriented graph, in order to model the hierarchy and the reading order of the different layout entities. In figure 5.2, this layout graph is projected on the document image: nodes are layout entities and edges model the relations between them. A membership relation is represented by a dashed arrow, while solid arrows represent the reading order. On this example, the document is made up of two pages, each page containing a page number and a sequence of sections, each section being made up of zero, one or many marginal annotations and a single body.

Text and layout are intrinsically linked: text recognition may help to label a layout entity, and vice versa. Therefore, we have turned toward the joint recognition of text and layout in a unique model. As shown on the right side, we chose the XML paradigm to generate a serialized representation of the document that is further used as the ground truth of a handwritten document. Notice that no physical information is encoded in the ground truth. Contrary to the standard DLA task, the proposed approach does not rely on segmentation labels of text regions such as bounding boxes for instance. Instead, the proposed model is able to provide transcriptions enriched with logical layout information, leading to structured transcriptions. It notably enables to reduce the need for costly segmentation annotations.

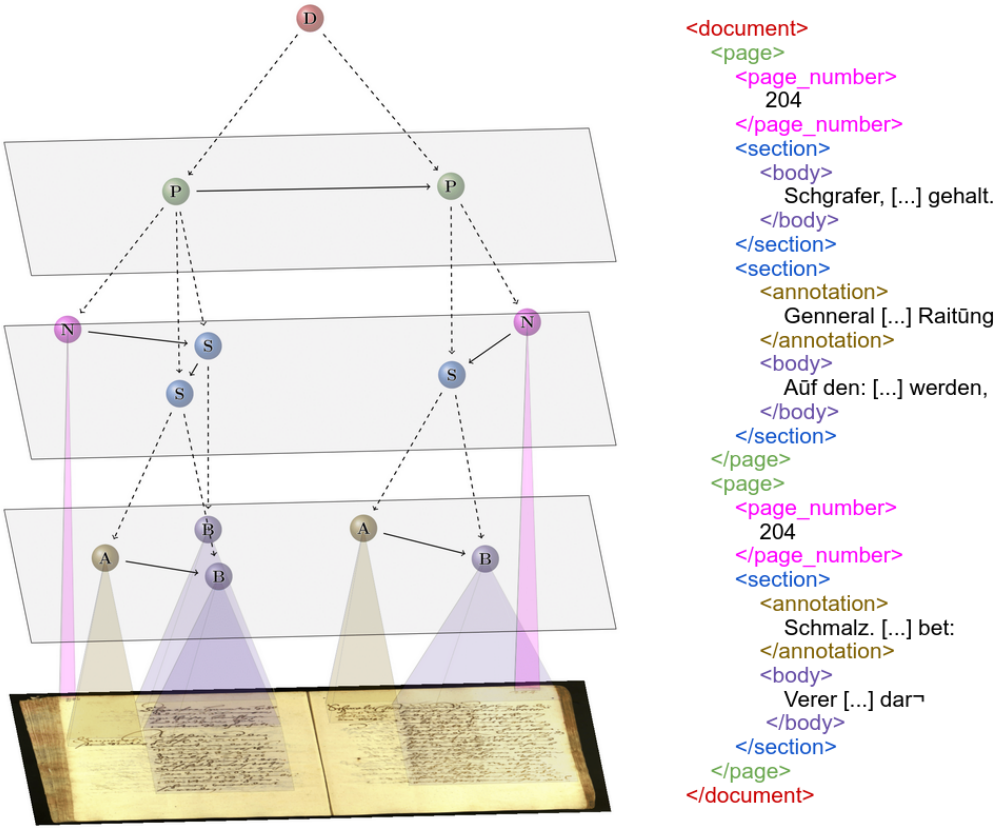


Figure 5.2: Left: document image with associated layout graph. Right: ground truth transcription including text and layout tokens.

The problem can be formalized as follows: the input is a raw document image $\boldsymbol{X}$ and the expected output is a sequence $\boldsymbol{y}$ of tokens, of length L_y. Tokens are grouped in the same dictionary $\mathcal{D} = \mathcal{A} \cup \mathcal{S} \cup \{< \text{eot} >\}$, where $\mathcal{A}$ are tokens of characters from a given alphabet, $\mathcal{S}$ are specific layout tokens and <eot> is a special end-of-transcription token. Layout tokens are pairs of tokens (begin and end) that tag sequences of character tokens as shown in Figure 5.2. As for characters, they vary according to the dataset used.

In the following, we present the DAN architecture and training strategy. We describe two new metrics we propose for the task of HDR. We provide an experimental study, including a visualization of the process, and a discussion of the model. We provide all source code and pre-trained model weights at `https://github.com/FactoDeepLearning/DAN`.

5.3.1 Architecture

We propose the DAN, an end-to-end encoder-decoder architecture that jointly recognizes both text and layout, from whole documents. We opted for an FCN as encoder since they are known to be efficient for feature extraction from images and can deal with input of variable sizes. For the decoder, we chose the transformer [65] because it is currently the state-of-the-art approach for many tasks involving the prediction of sequences of variable lengths.

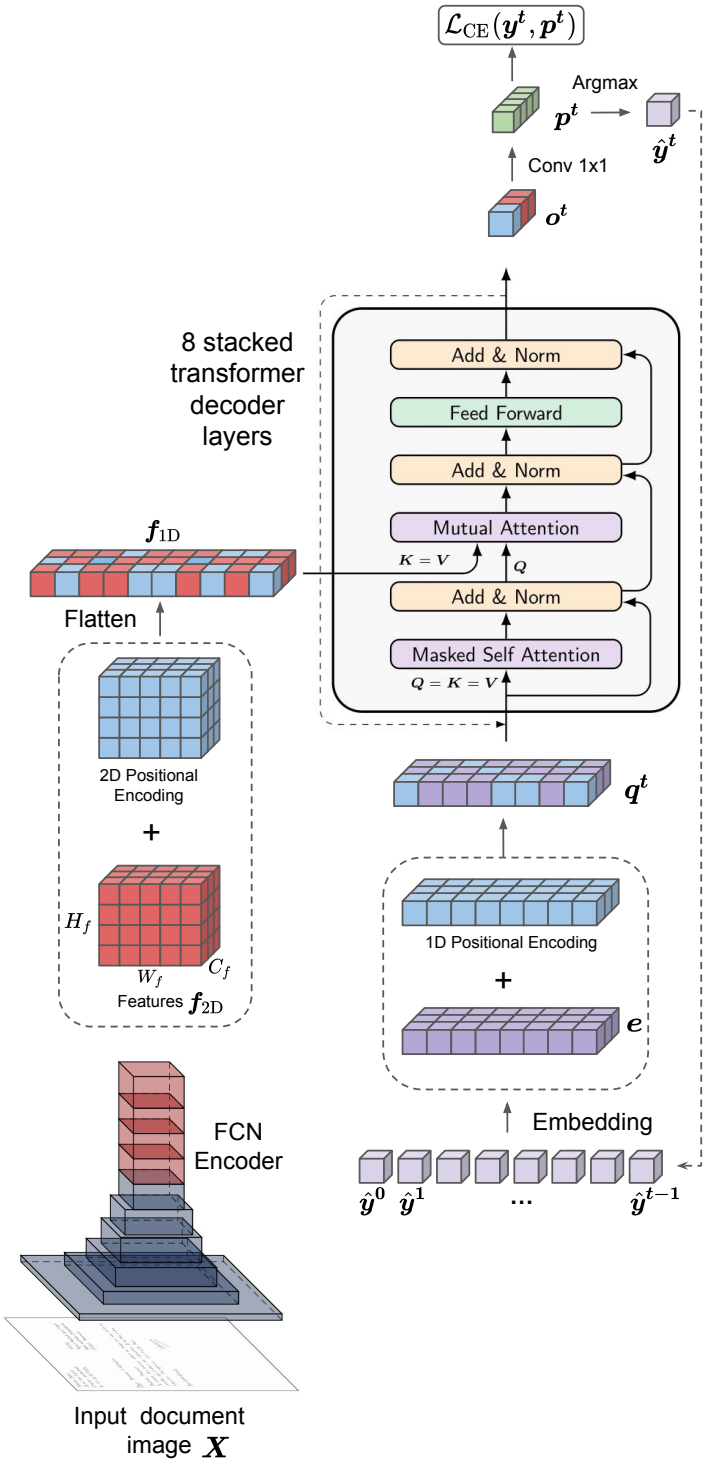


Figure 5.3: Overview of the DAN architecture (FCN+transformer decoder).

The DAN architecture is depicted in Figure 5.3. It is made up of an FCN encoder, to extracts 2D feature maps $\boldsymbol{f}_{2\text{D}}$ from an input document image $\boldsymbol{X}$. 2D positional encoding is added to these features in order to keep the spatial information, before being flattened into a

1D sequence of features $\boldsymbol{f}_{\text{1D}}$. This representation is computed only once and serves as input to the transformer decoder. The decoder follows a recurrent process at character-level: given the previously predicted tokens ($\hat{\boldsymbol{y}}^{\boldsymbol{0}}$, $\hat{\boldsymbol{y}}^{\boldsymbol{1}}$, ..., $\hat{\boldsymbol{y}}^{\boldsymbol{t-1}}$) and based on the computed features $\boldsymbol{f}_{\text{1D}}$, it outputs the next token probabilities $\boldsymbol{p}^{\boldsymbol{t}}$ for each token of $\mathcal{D}$. The final predicted token $\hat{\boldsymbol{y}}^{\boldsymbol{t}}$ is the one with the highest probability. The decoding process starts with an initial <sot> (start-of-transcription) token ($\hat{\boldsymbol{y}}^{\boldsymbol{0}} = <\text{sot}>$), and ends with the prediction of a special <eot> (end-of-transcription) token ($\hat{\boldsymbol{y}}^{\boldsymbol{L_y+1}} = <\text{eot}>$). We used the cross-entropy loss ($\mathcal{L}_{\text{CE}}$) for training.

We now describe the encoder and the decoder with more details.

5.3.1.1 Encoder

As in [72], we opted for an FCN encoder in order to better model the local dependencies since inputs are images. We used the FCN encoder of the Vertical Attention Network (VAN) [146], for many reasons. It achieves state-of-the art results for HTR at paragraph level on many public datasets: RIMES 2011, IAM and READ 2016. It can handle input of variable sizes. And it implies few parameters (1.7M) compared to other approaches.

The encoder takes as input a document image $\boldsymbol{X} \in \mathbb{R}^{H \times W \times C}$, with H, W and C being respectively the height, the width and the number of channels ($C = 3$ for an RGB image). It extracts some features maps for the whole document image: $\boldsymbol{f}_{\text{2D}} \in \mathbb{R}^{H_f \times W_f \times C_f}$ with $H_f = \frac{H}{32}$, $W_f = \frac{W}{8}$ and $C_f = 256$.

The original transformer architecture [65] is defined for 1D sequences. Since the inputs are 2D images, we replaced the 1D positional encoding by 2D positional encoding, as proposed in [72]. 2D positional encoding is defined as a fixed encoding based on sine and cosine functions with different frequencies (as in [65]); but instead of encoding a 1D position using all the channels, half is dedicated to vertical positional encoding and the other half to the horizontal positional encoding:

$$\begin{aligned} \text{PE}_{\text{2D}}(x, y, 2k) &= \sin(w_k \cdot y), \\ \text{PE}_{\text{2D}}(x, y, 2k+1) &= \cos(w_k \cdot y), \\ \text{PE}_{\text{2D}}(x, y, d_{\text{model}}/2 + 2k) &= \sin(w_k \cdot x), \\ \text{PE}_{\text{2D}}(x, y, d_{\text{model}}/2 + 2k + 1) &= \cos(w_k \cdot x), \\ \forall k \in [0, d_{\text{model}}/4]\,, & \end{aligned} \tag{5.1}$$

with

$$w_k = 1/10000^{2k/d_{\text{model}}}. \tag{5.2}$$

We set $d_{\text{model}} = C_f = 256$.

Features $\boldsymbol{f}_{\text{2D}}$ are summed with 2D positional encoding before being flattened for transformer decoder requirements:

$$\boldsymbol{f}_{\text{1D}_j} = \text{flatten}(\boldsymbol{f}_{\text{2D}_{x,y}} + \text{PE}_{\text{2D}}(x, y)), \tag{5.3}$$

with

$$j = yW_f + x. \tag{5.4}$$

5.3.1.2 Decoder

The decoder follows a recurrent process. At each iteration, it takes as input the flattened visual features $\boldsymbol{f}_{\text{1D}}$ and the previously predicted tokens ($\hat{\boldsymbol{y}}^{\boldsymbol{0}}$, ..., $\hat{\boldsymbol{y}}^{\boldsymbol{t-1}}$), and outputs the probabilities $\boldsymbol{p}^{\boldsymbol{t}}$ for each token in the dictionary $\mathcal{D}$. We used learned embeddings to convert the tokens to vectors $\boldsymbol{e}_{\hat{y}_i}$ of dimension d_{model}. Embeddings are summed with 1D positional encoding corresponding to the position of the predicted tokens in the predicted sequence, as in [65]:

$$\boldsymbol{q}_i^{\boldsymbol{t}} = \text{PE}_{\text{1D}}(i) + \boldsymbol{e}_{\hat{y}_i}, \tag{5.5}$$

with:

$$\begin{aligned} \text{PE}_{\text{1D}}(x, 2k) &= \sin(w_k \cdot x) \\ \text{PE}_{\text{1D}}(x, 2k+1) &= \cos(w_k \cdot x) \\ \forall k &\in [0, d_{\text{model}}/2]\,. \end{aligned} \tag{5.6}$$

The decoder is made up of a stack of 8 transformer decoder layers (as shown in Figure 5.3) followed by a convolutional layer with kernel 1×1 that computes the next token probabilities $\boldsymbol{p}^{\boldsymbol{t}}$. The transformer decoder layers are based on multi-head attention [65] mechanisms we denote as self attention and mutual attention. Self attention aims at modeling dependencies among the predicted sequence: it corresponds to multi-head attention where queries $\boldsymbol{Q}$, keys $\boldsymbol{K}$ and values $\boldsymbol{V}$ are from the same input. Mutual attention is used to extract visual information from the encoder ($\boldsymbol{K}$ and $\boldsymbol{V}$ are from $\boldsymbol{f}_{\text{1D}}$), based on $\boldsymbol{Q}$ which comes from the previous predictions. In other words, given the previous predictions, it indicates where should the model look at to predict the next token.

We used 8 decoder layers with dimension d_{model}, feed forward dimension d_{model}, 4 attention heads, ReLU activation and 10% of dropout. Self attention is causal since it is based on the previous predictions. As in [72], we used an attention window of length 100 for the self attention in order to reduce the computation time. It means that given an input sequence $\boldsymbol{s}$ of length L_{s}, the t^{th} output frame $\boldsymbol{o}^{\boldsymbol{t}}$ is computed over the range $[\boldsymbol{s}_a, \boldsymbol{s}_{t-1}]$ with $a = \max(0, t - 100)$. The process starts with a <sos> token and ends when a <eos> token is predicted or after L_{max} iterations. We set $L_{max} = 3000$ to match the datasets needs.

The whole model is made up of 7.6 M trainable parameters and is trained in an end-to-end fashion using the cross-entropy loss over the sequence of tokens:

$$\mathcal{L} = \sum_{t=1}^{L_y+1} \mathcal{L}_{\text{CE}}(\boldsymbol{y}^{\boldsymbol{t}}, \boldsymbol{p}^{\boldsymbol{t}}) \tag{5.7}$$

5.3.2 Training strategy

Training a deep attention-based neural network is difficult, especially when dealing with large inputs such as whole documents. The proposed training strategy is designed to improve the convergence while not relying on any segmentation label and dealing with few training data. It is performed in two steps:

- Pre-training: the aim is to learn the feature extraction part of the DAN. We trained a line-level OCR model on synthetic printed lines, and used it for transfer learning purposes for the DAN. We only used synthetic printed lines to avoid using segmentation labels which are costly annotations. Pre-training is carried out during 2 days with a mini-batch size of 16. Pre-processings, data augmentation and curriculum dropout are used during pre-training, as detailed afterwards.

- Training: the DAN is trained using teacher forcing (See section 5.3.2.5) to reduce the training time per epoch. It is trained on both real and synthetic documents. The idea is to learn the attention mechanism, *i. e.* the reading order, through the synthetic images. Indeed, printed text is easier to recognize than handwritten text and the DAN is pre-trained on printed text lines. Once the reading order is learned, it becomes easier to adapt to real-world images. This is motivated by the nature of the reading order: it is the same between printed and handwritten documents sharing the same layout.

  Following this idea, the model is first trained with 90% of synthetic documents during a curriculum phase to learn the reading order while using few real training samples. Then, this percentage is slowly decreased to reach 20% through the epochs, in order to fine-tune on the real samples while keeping some synthetic samples acting as unseen training data.

  Training is carried out during 4 days. We did not used mini-batch: training is carried out image per image. Pre-processings, data augmentation, curriculum strategies and post-processings are used during training, as described in the following.

5.3.2.1 Pre-training

Training deep attention-based models is difficult. It is beneficial to train part of the model beforehand whether it is the attention part or the feature extraction part, as shown in [146]. As in [146], we used a line-level pre-training strategy *i. e.* we first train a line-level OCR model on isolated line images using the Connectionist Temporal Classification (CTC) loss. This line-level OCR model is the same as the one used for the VAN and the Simple Predict & Align Network (SPAN) (Figure 4.9). However, contrary to [146, 145], we do not use real isolated lines (extracted with the bounding boxes annotations) but synthetic printed text lines, generated from the text line transcriptions only. The DAN is then trained using transfer learning using this model to initialize the weights of the encoder and of the decision layer (last convolutional layer of the decoder) with those of this line-level OCR model.

5.3.2.2 Curriculum strategies

We used two curriculum strategies to improve the convergence by slightly increasing the difficulty of the task during training.

A curriculum strategy was used for the generation of synthetic data during the training of the DAN. Instead of directly generating whole documents, we progressively increase the number of lines per page contained in the generated documents. We set the minimum number of lines to 1 and the maximum number of lines to $l_{\max}$, to fit the datasets properties. In addition, we also crop the synthetic document image under the lowest text line during this

curriculum stage. This way, we slightly increase both the length of the target sequence, through the number of text lines, and the input image size.

We used a second curriculum strategy regarding the dropout, as defined in [28]. It means that the dropout rate τ evolves during training:

$$\tau_t = (1 - \bar{\tau}) \exp(-\gamma t) + \bar{\tau}, \gamma > 0, \tag{5.8}$$

where $\bar{\tau}$ is the final dropout rate, t is the number of iterations (weight update) and $\gamma = \frac{1}{T}$, T being the total estimated number of weight updates during training. We set $T = 5 \times 10^4$.

5.3.2.3 Data Augmentation

We used a data augmentation strategy with a probability of 90%. This data augmentation strategy consists in applying some transformations, in random order, with a probability of 10% for each one. These data augmentation techniques are: resolution modification, perspective transformation, elastic distortion, dilation, erosion, color jittering, gaussian blur, gaussian noise and sharpening. Data augmentation is applied on both synthetic and real images.

5.3.2.4 Synthetic data

We generated synthetic printed lines for pre-training and synthetic printed documents for training. To this end, we arbitrarily chose a set of fonts $\mathcal{F}$ to introduce diversity in writing styles. We used these different fonts with various font sizes to bring more variability, making the model more robust. The original dataset $\mathcal{D}_{\text{doc}}$ is used to extract isolated text line transcriptions y_i associated to a layout class c_i, leading to a new dataset $\mathcal{D}_{\text{line}}$. Synthetic lines are generated on the fly during pre-training, by randomly selecting a text line transcription from $\mathcal{D}_{\text{line}}$.

While generating synthetic documents through learning have been studied in [153] for instance, here we focused on a rule-based approach for simplicity. Algorithm 4 details this generation process of synthetic documents. It is based on a style sheet s which defines the different layout entities (classes) in the document and a set of constraints on them, such as relative and absolute positioning rules. It also defines properties for each layout entity: maximum height or width in pixels, characters per line, line width or number of lines.

Synthetic documents are produced on the fly. A document image $\boldsymbol{X}$ is randomly chosen from the training dataset $\mathcal{D}_{\text{doc}}$ to get a realistic document shape, which is used as a template for a synthetic document $\boldsymbol{D}$ to be generated. Given the current curriculum number of lines per page l (between 1 and l_{max}), the actual number of lines for the current synthetic document l_{doc} is randomly chosen between 1 and l. Layout entities are generated one after the other until reaching l_{doc}: the layout class is chosen through "get_next_layout_class" based on the style sheet definition and the current state of $\boldsymbol{D}$. Given the remaining number of lines (l_{doc} - l_{current}), a random number of lines for the given entity l_{entity} is chosen in compliance with s. l_{entity} synthetic text line images are generated using a random font from $\mathcal{F}$ and a random text line from $\mathcal{D}_{\text{line}}$. These images are concatenated on the vertical axis, introducing some random indent spacing. The associated ground truth transcriptions are also concatenated in the same order. The generated layout entity is then placed into $\boldsymbol{D}$ in

Algorithm 4: Synthetic document generation.

input : original document image $\boldsymbol{X}$,
number of lines l_{doc},
style sheet s,
line-level dataset $\mathcal{D}_{\text{line}} = (\mathcal{Y}, \mathcal{C})$,
set of fonts $\mathcal{F}$.

output: synthetic document image $\boldsymbol{D}$,
ground truth y.

$y =$ " ";
$l_{\text{current}} = 0$;
$H, W = \text{size}(\boldsymbol{X})$;
$\boldsymbol{D} = \text{zeros}(H, W)$;
while $l_{\text{current}} < l_{\text{doc}}$ **do**
  $c = \text{get\_next\_layout\_class}(\boldsymbol{D}, s)$;
  $l_{\text{entity}} = \text{get\_num\_lines}(c, s, l_{\text{current}}, l_{\text{doc}})$;
  for $k = 1$ **to** l_{entity} **do**
    $y_k = \text{get\_random\_text}(\mathcal{D}_{\text{line}}, c)$;
    $f_k = \text{get\_random\_font}(\mathcal{F})$;
    $i_k = \text{generate\_text\_line\_image}(y_k, f_k)$;
  $i_{\text{entity}} = \text{merge\_text\_line\_images}(i_1, ..., i_{l_{\text{entity}}})$;
  $y_{\text{entity}} = \text{merge\_text\_line\_gt}(y_1, ..., y_{l_{\text{entity}}})$;
  $\boldsymbol{D} = \text{add\_layout\_entity}(\boldsymbol{D}, i_{\text{entity}}, s, c)$;
  $y = \text{add\_text}(y, y_{\text{entity}}, c)$;
  $l_{\text{current}} += l_{\text{entity}}$;
$\boldsymbol{D} = \text{crop\_under\_lowest\_entity}(\boldsymbol{D})$;

compliance with s. Ground truth transcription of each layout entity are concatenated by adding the corresponding layout tokens, leading to the ground truth of the whole synthetic document y. The document image is cropped under the lowest layout entity, as part of the curriculum strategy.

To sum up, we introduced variability in many points to generate different synthetic examples:

- different fonts and font sizes for the writing style.
- randomness and flexibility in the positioning constraints for the layout.
- random number of lines and mixed sample text lines for the content.
- cropping under the lowest layout entity for the image size.

5.3.2.5 Teacher forcing

We used teacher forcing at training time to parallelize the computations by predicting the whole sequence at once: the ground truth is used in place of the previously predicted tokens.

To make the DAN robust to errors occurring at prediction time, we introduced some errors in this sequence of pseudo previously predicted tokens. Some tokens are replaced by a random character or layout token. We found 20% to be a good error rate through experiments.

5.3.2.6 Pre-processings

To reduce the memory consumption, images are downscaled through a bi-linear interpolation to a 150 dpi resolution. Images are normalized (zero mean, unit variance) based on mean and variance computed on the images of the training set.

5.3.2.7 Post-processings

We used a rule-based post-processing to correct unpaired predicted layout tokens. It is essential to compute the metrics. Let us denote <X> a layout begin token and </X> its associated layout end token. The post-processing consists in a forward pass on the whole predicted sequence $\hat{\boldsymbol{y}}$ during which only tokens of layout are modified in order to have a coherent global structure. The main rules are:

- a missing end token is added when there are two successive begin tokens.
- isolated end tokens are removed.

For instance, omitting text prediction for simplicity, the prediction "<X> <Y> </Y> </Z>" becomes "<X> </X> <Y> </Y>".

In addition, the post processing ensures that the prediction is in accordance with the layout token grammar, *i. e.* the hierarchical relation between tokens are correct. It means that if a layout entity of class A can only be in an entity of class B, by definition, missing tokens are added. For example, the prediction "<A> </Y>" becomes "<B> <A> </A> </B>"

We used a second post-processing, for the text prediction: duplicated space characters are removed from the predicted sequence.

5.3.3 Metrics

The proposed approach aims at jointly recognizing both text and layout. While there exist well established metrics to measure the performance of both tasks independently, we are not aware of an adequate metric to evaluate both tasks when performed altogether. To our knowledge, there is no prior work handling such task. Therefore, we propose the evaluation of our approach using three different angles: the text recognition only, the layout recognition only and the joint recognition of both text and layout, using two new metrics named Layout Ordering Error Rate (LOER) and mAP_{CER}.

In the following we will take as example the following predicted sequence $\hat{\boldsymbol{y}}$, after post-processing:
"<X>text1</X><B><A>text2</A><A>text3</A></B>"

5.3.3.1 Evaluation of the text recognition

To evaluate the text recognition, all layout tokens are removed from the ground truth $\boldsymbol{y}$ and from the prediction $\hat{\boldsymbol{y}}$, leading to $\boldsymbol{y}^{\text{text}}$ and $\hat{\boldsymbol{y}}^{\text{text}}$, respectively.
Here, $\hat{\boldsymbol{y}}^{\text{text}}$ = "text1text2text3".

We used the standard Character Error Rate (CER) and the Word Error Rate (WER) to evaluate the performance of the text recognition. They are both computed as the sum of the Levenshtein distances (noted d_{lev}) between the ground truths and the predictions, at document level, normalized by the total length of the ground truths $\boldsymbol{y}^{\text{text}}_{\text{len}_i}$. For K examples in the dataset:

$$\text{CER} = \frac{\sum_{i=1}^{K} \text{d}_{\text{lev}}(\hat{\boldsymbol{y}}_i^{\text{text}}, \boldsymbol{y}_i^{\text{text}})}{\sum_{i=1}^{K} \boldsymbol{y}^{\text{text}}_{\text{len}_i}}. \tag{5.9}$$

WER is computed in the same way but at word level. Punctuation characters are considered as words.

One should note that, contrary to text line or paragraph recognition, the reading order is far more complicated to learn. An inversion in the reading order between two text blocks can severely impact the CER and WER values, even with correctly recognized text blocks.

5.3.3.2 Evaluation of the layout recognition

We cannot use existing DLA metrics, such as Intersection over Union (IoU), mean Average Precision (mAP) [81] or ZoneMap [154], to evaluate the layout recognition, because they are based on physical layout (segmentation) annotations.

We decided to model the layout as an oriented graph to take into account both the reading order and the hierarchical relations between layout entities. To evaluate the layout recognition, we introduce a new metric: the LOER. To this end, we associate to each ground truth and prediction a graph representation: $\boldsymbol{y}^{\text{graph}}$ and $\hat{\boldsymbol{y}}^{\text{graph}}$, as shown in Figure 5.4. We propose to generate this graph representation in two steps. First, we compute $\boldsymbol{y}^{\text{layout}}$ and $\hat{\boldsymbol{y}}^{\text{layout}}$, the ground truth and the prediction from which all but layout tokens are removed.
Here, $\hat{\boldsymbol{y}}^{\text{layout}}$ = "<X></X><B><A></A><A></A></B>".
Second, we map this sequence of layout tokens into a graph following the hierarchical rules of the datasets.

We designed LOER following the same paradigm as CER, adapting it to graph. It is computed as a Graph Edit Distance (GED), normalized by the number of nodes and edges in the ground truth. As shown in Figure 5.4, this graph can be represented by ordering the nodes with respect to a root D which represents the document. This way, the graph can be represented as a multi-level graph where the nodes are the different layout entities, the oriented edges between successive levels (dashed arrows) are their hierarchy and the oriented edges inside a same level (solid arrows) represent their reading order.

For K samples in the dataset, LOER is computed as the sum of the graph edit distances,

normalized by the sum of the number of edges n_{e} and nodes n_{n} in the ground truths:

$$\mathrm{LOER} = \frac{\sum_{i=1}^{K} \mathrm{GED}(\boldsymbol{y}_i^{\mathrm{graph}}, \hat{\boldsymbol{y}}_i^{\mathrm{graph}})}{\sum_{i=1}^{K} n_{\mathrm{e}_i} + n_{\mathrm{n}_i}}. \tag{5.10}$$

The graph edit distance is computed using a unit cost of edition whether it is for addition, removal or substitution and whether it is for edges or nodes. This computation becomes intractable in a reasonable running time for multiple pages. We circumvented this issue by assuming that the prediction of the page tokens was done in the right order. In this way, the GED of a document with several pages corresponds to the sum of the GED computed on the sub-graphs representing the isolated pages. Missing ground truth or prediction sub-graphs are compared to null graph.

Combining CER and LOER is not sufficient to evaluate the correct recognition of the document. As a matter of fact, one could reach 0% for both metrics by predicting first all the character tokens, and then all the layout tokens, each in the correct order. One misses the evaluation of the association between the layout tokens and their corresponding text parts.

5.3.3.3 Evaluation of joint text and layout recognition

We propose a second new metric, $\mathrm{mAP}_{\mathrm{CER}}$, to evaluate the joint recognition of both text and layout. It is based on the standard mAP score used for object detection approaches [80, 81]; but instead of using the IoU to consider a prediction as true or false, we use the CER. It is computed as follows:

- The predicted sequence $\hat{\boldsymbol{y}}$ and the ground truth sequence $\boldsymbol{y}$ are split into sub-sequences. These sub-sequences are extracted using the begin and end tokens of a same class. Then, they are grouped by classes into lists of sub-sequences. Results for our prediction example is given in Table 5.1.

  For each layout class, sub-sequences are ordered by their confidence score, computed as the mean between prediction probabilities associated to the begin and end tokens of this class (probabilities from $\boldsymbol{p}^{\boldsymbol{t}}$). A predicted sub-sequence is considered as true positive if the CER between this sub-sequence and a sub-sequence of the ground truth from the same class is under a given threshold. Otherwise, it is considered as false positive. When associated, the used sub-sequences are removed until there is no more sub-sequences in the ground truth or in the prediction.

- The average precision AP_c for a layout class c corresponds to the Area Under the precision-recall Curve (AUC). As in [81], it is computed as an approximation by summing the rectangular areas under the curve, formed by each modification of the precision p_n and recall r_n:

$$\mathrm{AP}_{\mathrm{CER}_c} = \sum (r_{n+1} - r_n) \cdot p_{\mathrm{interp}}(r_{n+1}), \tag{5.11}$$

Table 5.1: Sub-sequences are extracted, grouped and ordered by layout classes for mAP_{CER} computation. Left: tokens of the predicted sequence and associated confidence score. Consecutive text tokens are grouped for simplicity, their associated confidence score has been averaged. Right: sub-sequences are extracted by layout tokens and ordered given confidence score.

Token	Confidence
<X>	90%
text1	95%
</X>	70%
<B>	95%
<A>	82%
text2	73%
</A>	86%
<A>	80%
text3	89%
</A>	80%
</B>	75%

Score	Text
X	
80%	text1
A	
84%	text2
80%	text3
B	
85%	text2text3

with

$$p_{\text{interp}}(r_{n+1}) = \max_{\tilde{r}>r_{n+1}} p(\tilde{r}). \tag{5.12}$$

- The average precision is itself averaged for different CER thresholds, between $\theta_{\min} = 5\%$ and $\theta_{\max} = 50\%$ with a step $\Delta\theta = 5\%$, leading to 10 thresholds:

$$\text{AP}^{5:50:5}_{\text{CER}_c} = \frac{1}{10}\sum_{k=1}^{10} \text{AP}^{5k}_{\text{CER}_c}. \tag{5.13}$$

- The mean average precision for a document is then computed as a weighted sum over the different layout classes, weighted by the number of characters len_c in each class c:

$$\text{mAP}_{\text{CER}} = \frac{\sum_{c\in S} \text{AP}^{5:50:5}_{\text{CER}_c} \cdot \text{len}_c}{\sum_{c\in S} \text{len}_c} \tag{5.14}$$

- Finally, the global mAP for a set of documents is computed by averaging the mAP of the different documents, weighted by the number of characters in each document.

This way, the mAP_{CER} gives an idea of how well the text regions have been classified, based on the recognized text. It could not be used with the CER only because it does not evaluate the order of the classified text regions. Combining CER, LOER and mAP_{CER} enables to have a real estimation of the quality of the prediction for joint text and layout recognition.

However, one could argue that the layout recognition performance is biased by the post-processing we used. To evaluate the impact of the post-processing in the final results, we defined a dedicated metrics: the Post Processing Edition Rate (PPER). It is used to

understand how much of the layout recognition is due to the raw network prediction and how much is due to the post-processing. It is defined by the number of post-processing edition operations (addition or removal of layout tokens) n_{ppe}, normalized by the number of layout tokens in the ground truth $\boldsymbol{y}_{\text{len}}^{\text{layout}}$. For K examples:

$$\text{PPER} = \frac{\sum_{i=1}^{K} n_{\text{ppe}_i}}{\sum_{i=1}^{K} \boldsymbol{y}_{\text{len}_i}^{\text{layout}}}. \tag{5.15}$$

One should keep in mind that this metric only quantifies how much of the final layout prediction is due to post-processing through edition operations: these modifications can be either beneficial or unfavorable.

5.3.4 Experiments

This section is dedicated to the evaluation of the DAN for document recognition. We evaluate the DAN on the RIMES 2009 and READ 2016 datasets. We provide a visualization of the attention process and of the predictions. We also provide an ablation study to highlight the key components that made it possible to achieve these results.

5.3.4.1 Datasets

We evaluated the DAN on two public handwritten document datasets: RIMES [4, 1] and READ 2016 [3].

RIMES

We used the RIMES 2009 dataset, as detailed in Section 2.5.1. We used two kinds of annotation from this page-level dataset: transcription ground truth and layout analysis annotations. Text regions are classified among 7 classes: sender (S), recipient (R), date & location (W), subject (Y), opening (O), body (B) and PS & attachment (P). We used these classes and associated them to the corresponding text part: we do not use any positional ground truth to train the proposed model. We corrected 3 annotations where at least one text line was missing. Sometimes, the annotators proposed two options for a given word or expression, we systematically selected the first one in every cases.

Since there is no other work evaluating their model on this dataset, we also evaluate the performance of the DAN on the RIMES 2009 dataset at paragraph-level, as well as on the RIMES 2011 dataset [1] at paragraph and line levels. The idea is to compare the recognition performance between the different segmentation levels and with the state-of-the-art approaches which rely on pre-segmented input at line or paragraph level. One should notice that the paragraphs from RIMES 2011 are not extracted from RIMES 2009 so we cannot directly compare the results on both datasets, but it is the nearest comparison we can have.

READ 2016

We used the READ 2016 dataset at page-level. We also generated a double-page version of this same dataset by concatenating the images of successive pages. Only few pages of

the original dataset are not paired and have been removed. Based on their positions, we automatically added a class to each text regions among the 5 following classes: page (P), page number (N), body (B), annotation (A) and section (S) (group of linked annotation(s) and body). For comparison purposes, we also evaluate the DAN on the READ 2016 dataset at line and paragraph levels.

For both RIMES 2009 and READ 2016 datasets at page or double-page level, the reading order is deduced automatically from the paragraph positions as follows. For READ 2016, the reading order is from page to page: first, the page number, then section by section. Among a section, all annotations are read before the body. For RIMES 2009, text regions are read from top to bottom. If two text regions share a same vertical space, text regions are read from left to right. An example of each dataset is depicted in Figure 5.4. Text regions are represented by bounding boxes, colored given their class. We also represented the expected reading order as an oriented graph linking the different text regions. Both datasets have their own specificities. READ 2016 has a more regular layout compared to RIMES 2009 but it includes hierarchical layout tokens: bodies are included into sections, itself included into pages. In addition, it can contain multiple pages. RIMES 2009 provides more variability regarding the layout, but the layout tokens are sequential.

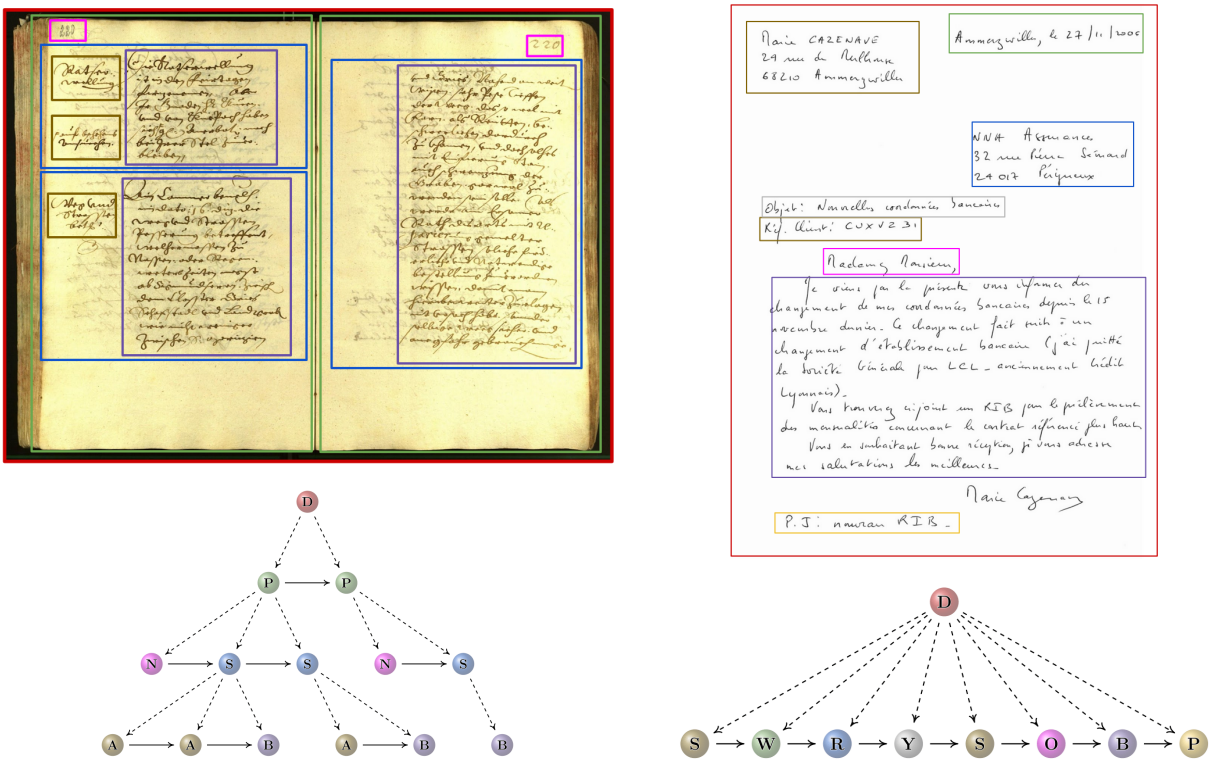


Figure 5.4: Images from READ 2016 and RIMES 2009 and associated layout graph annotation.

Datasets are split into training, validation and test sets, as detailed in Table 5.2. It corresponds to official or mainly used splits by the community. We also provide the number of characters for each dataset, as well as the number of layout tokens (two by class: begin

and end). We also provide this information at line and paragraph levels for comparison purposes.

Table 5.2: Datasets split in training, validation and test sets and associated number of tokens in their alphabet.

Dataset	Level	Training	Validation	Test	# char tokens	# layout tokens
RIMES 2011	Line	10,530	801	778	97	✗
	Paragraph	1,400	100	100	98	✗
RIMES 2009	Paragraph	5,875	540	559	108	✗
	Page	1,050	100	100	108	14
READ 2016	Line	8,367	1,043	1,140	88	✗
	Paragraph	1,602	182	199	89	✗
	Page	350	50	50	89	10
	Double page	169	24	24	89	10

5.3.4.2 Training details

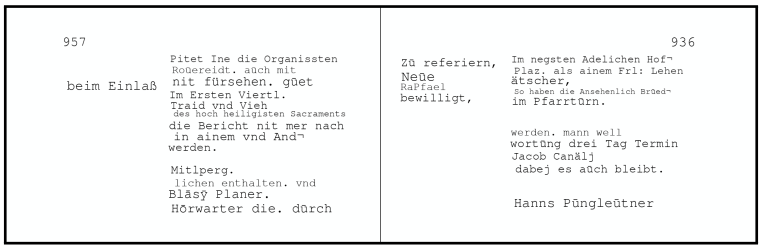
Pre-training and training are carried out with the same following configuration:

- Pytorch framework with automatic mixed precision.
- Training with a single GPU Tesla V100 (32Gb).
- Adam optimizer with an initial learning rate of 10^{-4}.
- We use exactly the same hyperparameters for both datasets.
- We do not use any external data, external language model nor lexicon constraints.

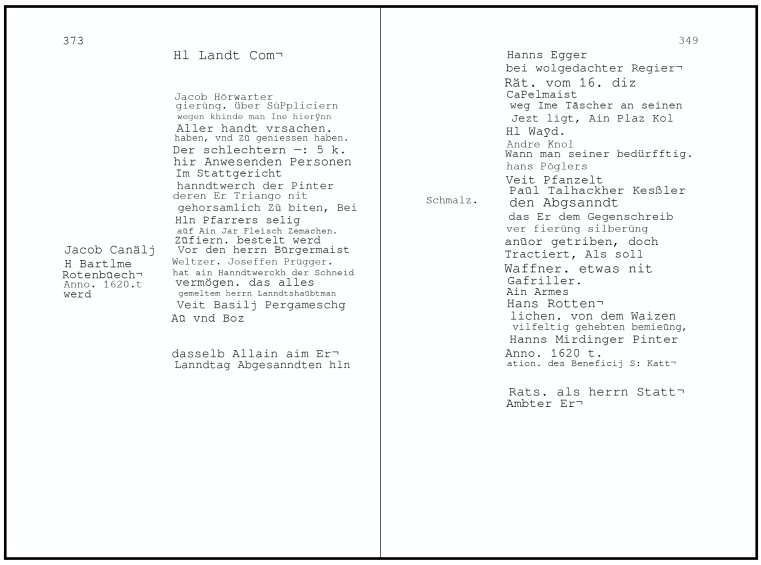
For the generation of synthetic documents, we set the maximum number of lines per page $l_{\max}$ to 30 for READ 2016 and to 40 for RIMES 2009 to match the dataset properties. Given a set of arbitrarily-chosen fonts, we only kept those for which all characters were supported, leading to 41 fonts for READ 2016 and 95 for RIMES 2009. The font lists are provided with the code for reproducibility purposes. Figure 5.5 illustrates the curriculum process for the generation of synthetic documents for the READ 2016 dataset at double-page level. As one can note, these synthetic documents are far from being visually realistic compared to the original dataset. This does not matter since the objective here is only to learn the reading order.

We also evaluate the DAN at paragraph and line levels for comparison purposes. For each training of a same dataset, at paragraph and page levels of RIMES 2009 for instance, the same pre-trained weights are used to initialize the model. Each evaluation corresponds to a specific training. Evaluation at paragraph or line levels corresponds to training only on synthetic and real paragraphs or lines, respectively.

(a) $l = 3$.



(b) $l = 15$.



(c) $l = l_{\max} = 30$ (end of curriculum stage, no crop).

Figure 5.5: Illustration of the curriculum learning strategy through the synthetic document image generation process for the READ 2016 dataset at double-page level.

5.3.4.3 Evaluation

To our knowledge, there is no work evaluating their system on the RIMES 2009 and READ 2016 datasets at page level. Comparison at paragraph and line levels are carried out with approaches under similar conditions *i. e.* without external data nor external language model. In Table 5.3, we present the evaluation of the DAN on the RIMES 2009 dataset at paragraph and page levels. One can notice that we reach very satisfying results at page level for both text and layout recognition with a CER of 4.54%, a WER of 11.85%, a LOER of 3.82% and a mAP_{CER} of 93.74%. The closest dataset with which we can compare is the RIMES 2011 dataset at paragraph level [1]. The DAN achieves new state-of-the-art results at paragraph level and competitive results at line level on this dataset. One should keep in mind that, as said previously, the comparison between RIMES 2009 and RIMES 2011 at paragraph level is not fair because RIMES 2011 only contains body images whose content seems easier to

recognize than that of RIMES 2009. Unique character sequences representing dates, postal codes, product and client references, or even proper nouns like names and places, are mainly in the other text regions. In addition, the body images from the RIMES 2011 dataset are not taken from the page images of RIMES 2009. This explains the CER difference between RIMES 2009 (5.46%) and RIMES 2011 (1.82%) at paragraph level. One can notice a CER improvement from line to paragraph level (for RIMES 2011) and from paragraph to page level (for RIMES 2009). This highlights the negative impact of using a prior segmentation step which is prone to annotation variations or errors.

Table 5.3: Evaluation of the DAN on the test set of the RIMES datasets and comparison with the state-of-the-art approaches.

Dataset	Approach	CER (%) ↓	WER (%) ↓	LOER (%) ↓	mAP_{CER} (%) ↑	PPER (%) ↓
RIMES 2011 [1]	**Line level**					
	[146] FCN	3.04	8.32	✗	✗	✗
	[124] CNN+BLSTM[a]	**2.3**	9.6	✗	✗	✗
	[155] DAN (FCN+transformer)[c]	2.63	**6.78**	✗	✗	✗
	Paragraph level					
	[145] SPAN (FCN)	4.17	15.61	✗	✗	✗
	[82] CNN+MDLSTM[b]	2.9	12.6	✗	✗	✗
	[146] VAN (FCN+LSTM)[b]	1.91	6.72	✗	✗	✗
	[155] DAN (FCN+transformer)[c]	**1.82**	**5.03**	✗	✗	✗
RIMES 2009 [4]	**Paragraph level**					
	[155] DAN (FCN+transformer)[c]	5.46	13.04	✗	✗	✗
	Page level					
	[155] DAN (FCN+transformer)[c]	4.54	11.85	3.82	93.74	1.45

[a] This work uses a slightly different split (10,203 for training, 1,130 for validation and 778 for test).
[b] with line-level attention.
[c] with character-level attention.

Table 5.4 provides an evaluation of the DAN on the READ 2016 dataset, at line, paragraph, single-page and double-page levels. As one can note, we achieve state-of-the-art results at line and paragraph levels. We also reach very interesting results whether it is at single-page or double-page level with a CER of 3.53% and 3.69%, respectively. It corresponds to slightly higher CER compared to the paragraph level (3.22%). One can note that the LOER and the mAP_{CER} are also satisfying, highlighting the good recognition of the layout. Moreover, these metrics are slightly better for the double-page level dataset. This could be explained by the higher necessity to understand the layout for complex samples. This is discussed in Section 5.3.4.5.

One should note that these CER values, for both RIMES 2009 and READ 2016 datasets, should not be compared directly to paragraph-level or line-level HTR approaches. Indeed, it is necessary to understand the impact of the reading order. CER is computed based on the edit distance between two one-dimensional sequences. This way, if the reading order is wrong, it impacts severely the CER. For instance, if the sender coordinates are read before the recipient coordinates while it is the opposite in the ground truth, the edit distance will be very important even if the text is well recognized. It means that part of this CER is due to a wrong reading order and not to a wrong text recognition. However, mAP_{CER} would not be impacted: it is invariant to the reading order between the different text regions.

Table 5.4: Evaluation of the DAN on the test set of the READ 2016 dataset and comparison with the state-of-the-art approaches

Approach	CER (%) ↓	WER (%) ↓	LOER (%) ↓	mAP_{CER} (%) ↑	PPER (%) ↓
Line level					
[131] CNN+BLSTM[a]	4.66	✗	✗	✗	✗
[3] CNN+RNN	5.1	21.1	✗	✗	✗
[146] VAN (FCN+LSTM)[b]	**4.10**	**16.29**	✗	✗	✗
[155] DAN (FCN+transformer)[a]	**4.10**	17.64	✗	✗	✗
Paragraph level					
[145] SPAN (FCN)	6.20	25.69	✗	✗	✗
[146] VAN (FCN+LSTM)[b]	3.59	13.94	✗	✗	✗
[155] DAN (FCN+transformer)[a]	**3.22**	**13.63**	✗	✗	✗
Single-page level					
[155] DAN (FCN+transformer)[a]	3.53	13.33	5.94	92.57	0.15
Double-page level					
[155] DAN (FCN+transformer)[a]	3.69	14.20	4.60	93.92	1.37

[a] with character-level attention.
[b] with line-level attention.

For both datasets, the PPER metric is very low. It indicates that the good results obtained for the layout recognition are mainly due to the DAN itself and not to the post-processing stage.

We now focus on the mAP_{CER} metric with the READ 2016 dataset at double-page level. In Table 5.5, we detailed this metric for each layout class and for each threshold of CER. As one can notice, pages, page numbers, sections and bodies are very well recognized with at least 93% for a CER as of 10%. The DAN has more difficulty with the annotations, with an average of 81.64%. We assume that is due to two main points: it is the layout entity with the more variability. There can be zero, one or multiple annotations per body. In addition, they can be placed wherever along the body, from its beginning to its end. Second point is about the length of the annotations: they are very shorter than bodies. This way, only few errors can lead to an important CER increase, leading to lower average precision.

Table 5.5: mAP_{CER} detailed for each class and each CER threshold, for the READ 2016 double-page dataset.

	$AP^{5:50:5}_{CER}$	AP^{5}_{CER}	AP^{10}_{CER}	AP^{15}_{CER}	AP^{20}_{CER}	AP^{25}_{CER}	AP^{30}_{CER}	AP^{35}_{CER}	AP^{40}_{CER}	AP^{45}_{CER}	AP^{50}_{CER}
Page (P)	97.19	78.12	93.75	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00
Page number (N)	98.12	96.88	96.88	96.88	96.88	96.88	96.88	100.00	100.00	100.00	100.00
Section (S)	96.52	78.04	93.33	97.78	98.89	98.89	98.89	99.84	99.84	99.84	99.84
Annotation (A)	81.64	47.78	66.27	76.31	85.78	89.71	89.71	89.71	89.71	89.71	91.67
Body (B)	94.33	84.63	93.15	95.19	95.19	95.19	95.19	96.19	96.19	96.19	96.19

5.3.4.4 Visualization

A visualization of the prediction for a test sample of the RIMES 2009 dataset is depicted in Figure 5.6 [a]. On the left, attention weights of the last mutual attention layer are projected on the input image. The colors depend on the last predicted layout token. For visibility, the intensity of the colors is encoded for attention values between 0.02 and 0.25. The text prediction is added in red line by line, under the associated text line. The corresponding layout graph is depicted on the right, with each node corresponding to a text region in the input image. As one can note, even if the DAN is not trained using any segmentation label, it performs a kind of implicit segmentation in its process, which can be globally retrieve through the attention weights. As one can notice, the DAN performs a document recognition: it recognizes both text and layout.

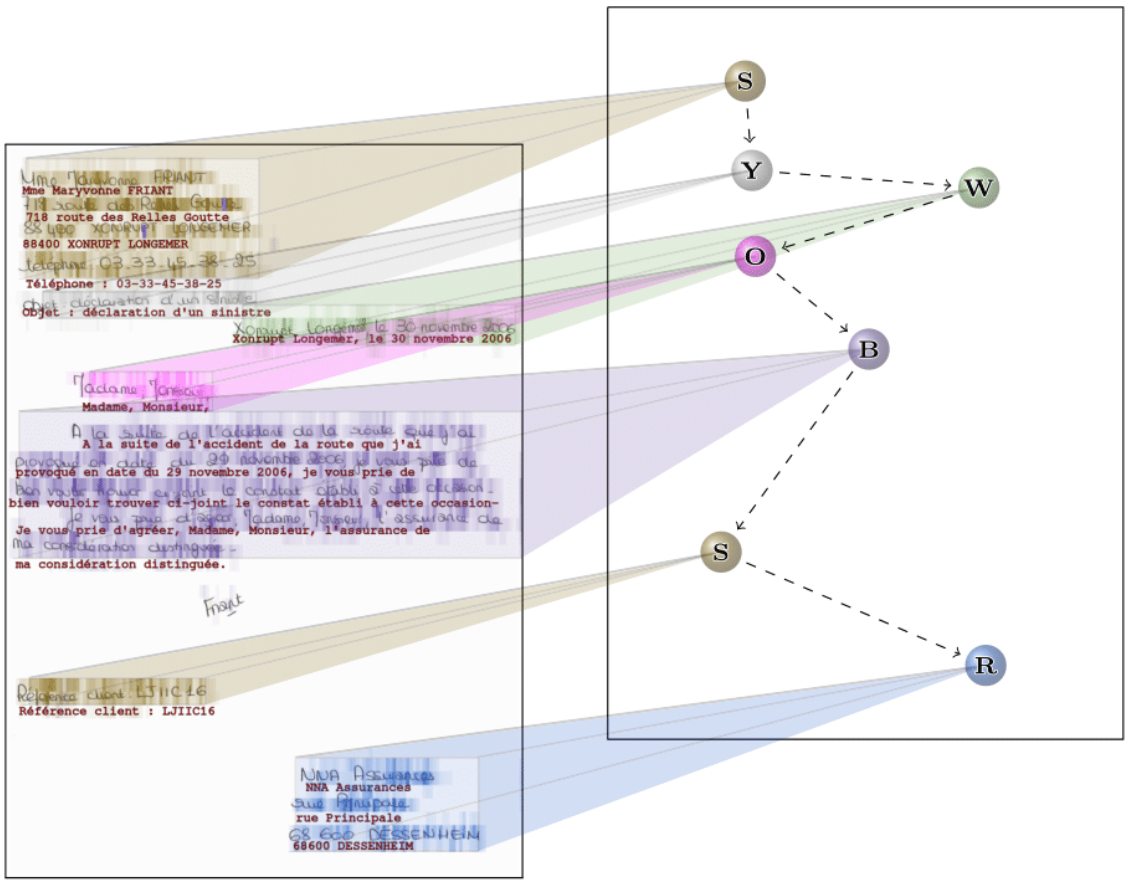
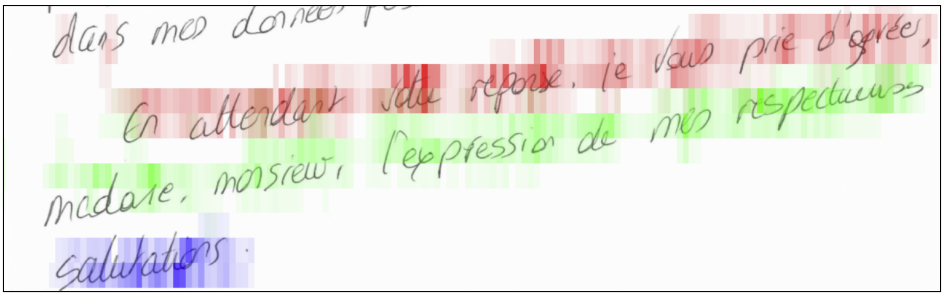


Figure 5.6: Visualization of the prediction on a RIMES 2009 test sample.

In addition, the DAN is able to deal with slanted lines through its character-level attention mechanism. A prediction visualization for such example is depicted in Figure 5.7. This time, we used one color per line for visibility. As one can note, the attention mechanism correctly follows the slope of the lines, leading to no error in prediction. This is an improvement

[a] Full demo at `https://www.youtube.com/watch?v=HrrUsQfW66E`



Prediction: "En attendant votre réponse, je vous prie d'agréer, Madame, Monsieur, l'expression de mes respectueuses salutations."

Figure 5.7: Attention weights visualization. Focus on slanted lines of a validation sample of the RIMES 2009 dataset.

compared to approaches based on line-level attention, which cannot handle such close and slanted lines.

5.3.4.5 Ablation study

We provide an extensive ablation study in Table 5.6. The evaluation is carried out on the test set of the RIMES 2009 dataset at page level and the READ 2016 dataset at double-page level, after a 2-day training. We evaluated the proposed training approach through the independent removal of each of the training component we used.

Table 5.6: Ablation study of the DAN on the RIMES 2009 and READ 2016 datasets. Results are given for the test sets for a 2-day training. All metrics are given in percentages.

	RIMES 2009 (single-page)				READ 2016 (double-page)			
	CER ↓	WER ↓	LOER ↓	mAP_{CER} ↑	CER ↓	WER ↓	LOER ↓	mAP_{CER} ↑
Baseline	5.72	13.05	**4.18**	**92.86**	**4.36**	16.55	**3.80**	**93.78**
(1) No synthetic data	8.26	16.45	8.18	86.34	80.75	95.65	36.77	0.13
(2) No curriculum for syn. data	7.59	16.48	6.63	88.92	81.31	92.66	21.12	0.24
(3) No crop in curr. for syn. data	5.84	13.73	4.42	91.94	99.98	100.00	85.77	0.00
(4) No data augmentation	7.08	15.54	4.78	91.65	100.00	100.00	43.45	0.00
(5) No curriculum dropout	5.83	14.41	4.36	92.09	4.59	**16.53**	4.98	92.97
(6) No error in teacher forcing	8.09	15.12	5.91	89.24	5.44	18.67	4.98	90.99
(7) No layout recognition	**5.30**	**12.46**	✗	✗	100.00	100.00	✗	✗
(8) No pre-training	71.42	87.48	18.46	12.72	5.52	19.62	6.29	89.38
(9) No 1D positional encoding	8.04	16.93	5.73	90.65	5.30	18.64	6.29	90.51
(10) No 2D positional encoding	12.43	20.83	8.42	89.81	70.84	93.73	23.94	20.04

Experiments (1) to (3) are dedicated to the generation of synthetic documents at training time. In (1), we do not generate synthetic documents: the model is only trained on real documents. In (2), the synthetic documents do not follow a curriculum approach: the maximum number of lines per document is directly set to its upper bound ($l = l_{\max}$). In (3), the synthetic document images are not cropped below the lowest text line during the curriculum phase: they preserve the original image height. As one can note, the removal of either of these three aspects prevents the DAN from converging, on the READ 2016 dataset. It also leads to poorer results on RIMES 2009, for each metric. One can note an increase of

0.12 points for the CER in the best case and of 2.54 points in the worst case. The mAP_{CER} is decreased by 0.92 to 6.52 points. The removal of the data augmentation strategy during both pre-training and training, experimented in (4), leads to similar results.

In (5), we do not use curriculum dropout strategy neither during pre-training nor during training. This time, it does not prevent the DAN from converging on the READ 2016 datasets. Except for the WER for READ 2016, the results are slightly worse. In experiment (6), we do not introduce any error in the teacher forcing strategy: we preserve the ground truth. The introduction of errors improves the results. We assume that training the model with errors helps it to deal with the errors made at prediction time.

The layout tokens are removed from the ground truth in (7) leading to text recognition only. As one can notice, it leads to lower CER and WER for the RIMES 2009 dataset but prevents the convergence for the READ 2016 dataset. This can be explained by the fact that the reading order is easier for RIMES 2009 than for READ 2016, especially at double-page level. We assume that recognizing the different layout entities enables to understand the spatial relation between them and helps to learn the reading order. Annotations are always at the left of a body: after the prediction of an </annotation> token, the attention must be focused righter to predict a <body> token.

In (8), the DAN is trained from scratch, without transfer learning from a prior pre-training step. Results are dramatically worse for the RIMES 2009 dataset. We assume that this is due to its irregular layout. Compared to READ 2016, RIMES 2009 provides simpler reading orders but with more variability in the layout. This way, we assume that for READ 2016, the reading order can be learned jointly with the feature extraction part, directly during the curriculum step. For RIMES 2009, the layout variability slows down the learning of the feature extraction part, leading to slower convergence.

Finally, in (9) and (10), we evaluate the importance of the 1D and 2D positional encoding components. As one can notice, they both enables to improve the results, especially the 2D positional encoding. This is especially true for the READ 2016 dataset. We assume this is due to the double-page nature of this dataset which leads to a more complex layout, resulting in more important jumps from one character prediction to another. For example, from the last character of the left page to the first one of the right page.

5.3.5 Discussion

We proposed a new end-to-end segmentation-free architecture for the task of Handwritten Document Recognition. To our knowledge, it is the first end-to-end approach that can deal with whole documents, recognizing both text and layout. As we have seen, the DAN reaches great results on both text and layout recognition whether it is at single-page or double-page level, on the RIMES 2009 and READ 2016 datasets. Indeed, we showed that the DAN is robust to the dataset varieties: we used the same hyperparameters with two totally different datasets in terms of layout, language, color encoding as well as number of training samples.

We proposed an efficient training strategy and we highlighted its impact on an extensive ablation study. This training strategy is based on synthetic line and document generation using digital fonts, to overcome the lack of training data. Pre-training is carried out on synthetic text lines only, avoiding to use any segmentation annotations: this is a great advantage compared to state-of-the-art approaches. We showed that even with complex

handwritings such as those of the READ 2016 dataset, for which the fonts aspect is very far from the original writings, this approach still remains effective.

However, it should be noted that the datasets are rather specific: letters for RIMES and historical book pages for READ 2016. The obtained results are very dependent on the quality of the synthetic data *i. e.* they must be close to the original dataset, notably in terms of layout. We used a reading order automatically generated from pixel positions of the different text regions. This is effective because we used datasets with rather regular layouts. This would be more complex for datasets of documents with more heterogeneous layouts. However, we assume that this approach could be generalized to heterogeneous documents without any problem by labeling a coherent reading order from one example to another. The main issue is the lack of large-scale public datasets for handwritten document recognition. We hope that this contribution will lead to the production of datasets at lower cost: the DAN only needs the ordered transcription annotation and layout tags, without the need for any segmentation annotation.

The DAN learns the reading order through the transcription annotations. We observed an interesting effect linked to this. In the specific case of READ 2016 at double-page level, the page number is identical for both left and right pages. The DAN focuses on the same area to predict both numbers. Technically, this does not impact the performance, but it shows that the network has not fully learned the concept of reading. We assumed that this phenomenon would disappear by learning on heterogeneous layouts.

The DAN is based on a recurrent process. This is not a problem at training time since computation are parallelized through teacher forcing. However, this recurrence issue is significant at prediction time: it grows linearly with the number of tokens to be predicted. This way, the average prediction time for a test sample is 5.8s for RIMES and 4.3s for READ 2016 at single-page level. It becomes 9.7s for READ 2016 at double-page level. This can be an obstacle for an industrial application. We aim at reducing this prediction time in future works.

5.4 Conclusion

We proposed to handle the task of HDR which corresponds to the joint recognition of text and layout. This is a new step toward the global understanding of whole handwritten documents. Moreover, we showed that this additional layout recognition enables to improve the text recognition performance. We assume that this is due to a better understanding of the reading order through the layout entity recognition. We proposed the first model to tackle this task, the DAN, as well as metrics to evaluate the associated performance. We reached very interesting results on two different but relatively homogeneous datasets. These results are comparable to those obtained at paragraph level, without using any segmentation labels, which is very encouraging. In future works, it would be interesting to go a step further, by recognizing handwritten documents with heterogeneous sizes and layouts. The prediction time, which grows linearly with the length of the character sequence, is another area for improvement.

Chapter 6

General conclusion and perspectives

In this thesis, we focused on the problem of Handwritten Text Recognition (HTR). As we have seen, this image-to-sequence problem is a difficult task, which implies many challenges. The variety of the writing styles, the variable length of the character sequences from one document to another, which can be relatively large, as well as the involved 2D-to-1D transformation between the input image and the expected sequence of characters are the most important ones. HTR has long been relying on a prior segmentation step at character, word or line level, and more recently at paragraph level.

We believe that the line-level approach is quite mature. Numerous architectures have been proposed, studied and successfully applied for this task and we showed that Fully Convolutional Network (FCN) architectures can compete with recurrent ones. In addition, they can easily be used for transfer learning purposes for paragraph-level or document-level HTR models. We assume that the performances of the current line-level systems are mainly limited by the lack of annotated data and we believe that research should now focus on multi-line approaches.

To this end, we proposed the Simple Predict & Align Network (SPAN) and demonstrated that paragraph-level HTR can be handled in such a simple way. This is the first one-shot approach for paragraph-level HTR that can deal with inputs of variable sizes. We proposed the Vertical Attention Network (VAN) which reached state-of-the-art results on three public datasets, bridging the gap between paragraph-level and line-level recognition performances.

However, these paragraph-level approaches still have some drawbacks:

- As for the other paragraph-level approaches proposed in the literature, the SPAN and the VAN are pre-trained on isolated text lines, using additional line-level segmentation annotations.
- They rely on prior segmentation and ordering steps, implying cumulative errors between each stage. However, they are evaluated on manually segmented images: the segmentation stage is almost never taken into account by the metrics.

We showed that the first issue can be alleviated by designing a synthetic data generation strategy. Indeed, this task enables us to generate printed documents in a rather easy way and which can be used for line-level pre-training. We argue that processing whole documents in an end-to-end way is the solution to consider to solve the second issue. Not only does

it avoid error accumulation, but it enables to take advantage of layout information for text recognition, and vice versa. To this aim, we proposed the Document Attention Network (DAN) as the first end-to-end model performing the task of Handwritten Document Recognition (HDR), recognizing both text and layout. We showed that it was feasible to process whole documents with rather complex layouts *i. e.* two-column documents, one to two pages long, while reaching competitive results, and without relying on any physical segmentation annotations.

All the approaches proposed in this thesis represent interesting steps towards the understanding of documents. However, document understanding field is still in its infancy and there are still many areas of research to be explored.

Perspectives

Improving the recognition. Whether it is at line, paragraph or document level, there seems to be only little improvement in terms of Character Error Rate (CER) these last years. If we focus on the IAM dataset for example, the approaches of the literature generally reach a CER between 4% and 5% [82, 83, 144, 146]. The only works that reach CER of less than 4% [126, 122], down to 2.75% [137], use additional external data and/or an external language model (character and/or word N-gram). On the one hand, designing an external language model implies a multi-step approach; it requires additional training which can be long and whose hyperparameters can be complex to fine-tune. On the other hand, the external data used are mainly private and it is very costly to annotate new training data. In this respect, although the ideal would be to have an large annotated dataset of reference, like ImageNet, it would be interesting to focus on annotation-free approaches to improve the recognition performance.

As we said, this stagnation of the CER can be observed at any level (line, paragraph, document). This leads us to think that the feature extraction part (encoder) would be the bottleneck regarding the architectures, rather than the decoder part. In this regard, it would be interesting to study the novel promising image-to-sequence architectures such as the Vision Transformer [73, 74] in the context of HTR. Moreover, this transformer-based architecture includes a kind of internal language model through its use of self-attention mechanism which could compensate the use of an external language model. However, the drawback of the Vision Transformer is its need for large training datasets so as to reach interesting results. As for all approaches based on neural networks, the size of the dataset is a key issue, especially for this type of architecture.

When proposing the DAN, we showed that using synthetic printed documents is efficient to train the attention part of the model but we assume that it is quite limited for the learning of the feature extraction part in the context of handwritten text recognition. Self-supervised learning is an emerging field of research which aims at learning from raw inputs, without using any manual annotation. This way, it would be interesting to focus on self-supervised learning approaches to take advantage of the astronomical number of handwritten documents that exist on earth and which are not annotated. More specifically, contrastive learning is a category of self-supervised learning methods that seems promising [156, 157, 158, 159]. To our knowledge, only one work focused on self-supervised contrastive learning approach for

HTR, at line level [160]. However, the authors of [161] reaches interesting results by applying a self-supervised approach to the Vision Transformer architecture.

Toward the recognition of heterogeneous and complex handwritten documents. We proposed the DAN as the first end-to-end approach for HDR. However, we limit ourselves to the evaluation of the DAN on homogeneous datasets (pages from a same book for READ 2016 and letters for RIMES 2009). Moving from the recognition of homogeneous to heterogeneous documents raises several issues. The layout entities can be very related to the nature of the document: question/answer fields are typical of forms, and the mention of a subject is indicative of a letter. The high variety of existing documents can lead to a relatively high number of classes to be recognized. In addition, if we focus on the DAN architecture, it requires to serialize the annotation of the document *i. e.* to set a reading order between the different text regions. Given the layout variability from one document to another, it is necessary to annotate the reading order so as to obtain a global coherence, no matter the nature of the document, which can be difficult.

Moreover, for documents with complex layouts such as schemes or maps, there is no human consensus about a unique reading order. This way, it would be interesting to focus on the training strategy, especially on the loss, to design a way to consider multiple reading orders as valid. The same goes for the metrics in order to have a coherent evaluation of the performance. In this the respect, the standard reference CER metric cannot be used in this context. The mAP_{cer} we proposed is a first proposition in that sense.

Another drawback of the DAN we mentioned is about its prediction time, which linearly grows with the length of the sequence of characters to be recognized. It results in another point of improvement. It would be interesting to study the use of multiple reading heads that would advance in parallel on several parts of the input document in order to minimize the prediction time.

Given the interesting results obtained with the Transformer architecture in many tasks, we assume that it would be possible to recognize more than just text and layout. For example, Transformers have been successfully applied for named entity recognition [147] or non-textual item detection [72], in addition to the text recognition. Transformers has also been studied for handwritten mathematical expression recognition for instance [162]. This way, we think it would be possible to train a model to recognize all of these items in an end-to-end way: text, layout, mathematical expression, and others such as tables. But as for standard handwritten documents, the main issue is the lack of datasets. We hope that the contributions of this thesis will encourage the community to create datasets for the end-to-end recognition of whole documents.

Handwritten text recognition has been studied for decades and we think it is now time to consider the task of document recognition in a more general way. We assume the coming years will be very interesting from this point of view.

List of Publications

Denis Coquenet, Clément Chatelain, and Thierry Paquet. "DAN: a Segmentation-free Document Attention Network for Handwritten Document Recognition". In: *Under review* (2022). URL: https://arxiv.org/abs/2203.12273

Denis Coquenet, Clément Chatelain, and Thierry Paquet. "End-to-end Handwritten Paragraph Text Recognition Using a Vertical Attention Network". In: *Transactions on Pattern Analysis and Machine Intelligence (TPAMI)* (2022)

Denis Coquenet, Clément Chatelain, and Thierry Paquet. "SPAN: a Simple Predict & Align Network for Handwritten Paragraph Recognition". In: *International Conference on Document Analysis and Recognition (ICDAR)*. vol. 12823. 2021, pp. 70–84

Denis Coquenet, Clément Chatelain, and Thierry Paquet. "Recurrence-free unconstrained handwritten text recognition using gated fully convolutional network". In: *17th International Conference on Frontiers in Handwriting Recognition (ICFHR)*. 2020, pp. 19–24

Denis Coquenet et al. "Have convolutions already made recurrence obsolete for unconstrained handwritten text recognition ?" In: *Workshop on Machine Learning (WML@ICDAR)*. 2019, pp. 65–70

List of Communications

Denis Coquenet, Clément Chatelain, and Thierry Paquet. "End-to-end Handwritten Paragraph Text Recognition Using a Vertical Attention Network". In: *Journée scientifique organisé par Normastic, Normaths, pôle SN et la graduate school MinMacs*. 2022 (Poster presentation)

Denis Coquenet, Clément Chatelain, and Thierry Paquet. "Handwritten text recognition: from isolated text lines to whole documents". In: *Journées francophones des jeunes chercheurs en vision par ordinateur (ORASIS)*. 2021 (Oral presentation)

Denis Coquenet, Clément Chatelain, and Thierry Paquet. "Intelligence artificielle : quand la machine apprend à lire". In: *Fête de la Science, UFR Sciences et Techniques de Rouen*. 2021 (Oral presentation)

Denis Coquenet, Clément Chatelain, and Thierry Paquet. "End-to-end Handwritten Paragraph Text Recognition Using a Vertical Attention Network". In: *Symposium International Francophone sur l'Ecrit et le Document (SIFED)*. 2021 (Oral presentation)

Denis Coquenet, Clément Chatelain, and Thierry Paquet. "Recurrence-free unconstrained handwritten text recognition using gated fully convolutional network". In: *4th Data Science Summer School (DS3)*. 2021 (Poster presentation)

Denis Coquenet, Clément Chatelain, and Thierry Paquet. "Recurrence-free unconstrained handwritten text recognition using gated fully convolutional network". In: *Symposium International Francophone sur l'Ecrit et le Document (SIFED)*. 2020 (Oral presentation)

Denis Coquenet et al. "Importance of recurrent layers for unconstrained handwriting recognition". In: *Symposium International Francophone sur l'Ecrit et le Document (SIFED)*. 2019 (Oral presentation)

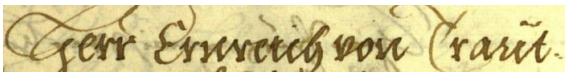
Appendix A

Data augmentation

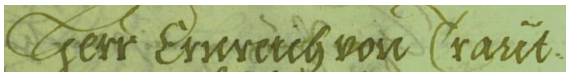
Data augmentation is a way to artificially augment the number of training data. It is based on transformation techniques which alter an original training sample.

The transformations used are subject to only one constraint: the relevant information must be preserved so that the ground truth remains unchanged. In the case of HTR, the characters must remain in the same order and recognizable. Figure A.1 illustrates the different data augmentation techniques used in this thesis.

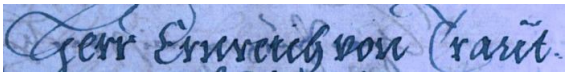
Some of these transformations are based on the modification of color properties such as hue, contrast, saturation and brightness; it is generally referred to as color jittering. It can be seen as adding artificial variation in terms of digitization process and conditions. Noise can also be added to make the character recognition more robust through gaussian blurring or gaussian noise addition. One of the main difficult aspects in HTR is the variation of writing style from one person to another. Many techniques can be used to simulate writing style variety: dilation, erosion and sharpening for the stroke width, resolution modification and zoom for the writing size, perspective transformation for slanted writing and elastic distortion for character shape. We combined these different data augmentation techniques in order to bring even more variability.



(a) Original.



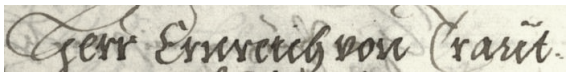
(b) Color jittering.



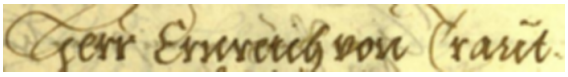
(c) Hue.

(d) Contrast.

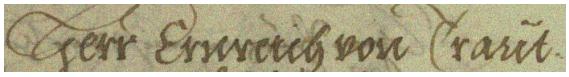
(e) Brightness.



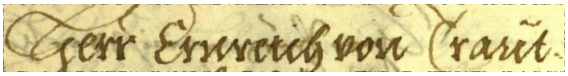
(f) Saturation.



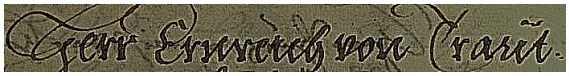
(g) Gaussian Blur.



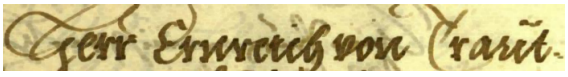
(h) Gaussian Noise.



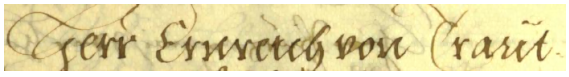
(i) Elastic distortion.



(j) Sharpening.

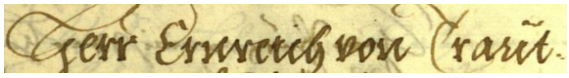


(k) Dilation.

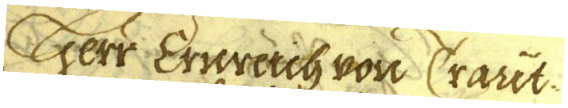


(l) Erosion.

(m) Zoom.



(n) Resolution modification.



(o) Rotation.

(p) Perspective transformation.

Figure A.1: Data augmentation techniques for HTR.

Bibliography

[1] Emmanuele Grosicki and Haikal El Abed. "ICDAR 2011 - French Handwriting Recognition Competition". In: *International Conference on Document Analysis and Recognition (ICDAR)*. 2011, pp. 1459–1463.

[2] Urs-Viktor Marti and Horst Bunke. "The IAM-database: an English sentence database for offline handwriting recognition". In: *International Journal on Document Analysis and Recognition (IJDAR)* 5.1 (2002), pp. 39–46.

[3] Joan-Andreu Sánchez et al. "ICFHR2016 Competition on Handwritten Text Recognition on the READ Dataset". In: *15th International Conference on Frontiers in Handwriting Recognition (ICFHR)*. 2016, pp. 630–635.

[4] Emmanuele Grosicki et al. "Results of the RIMES Evaluation Campaign for Handwritten Mail Processing". In: *10th International Conference on Document Analysis and Recognition (ICDAR)*. 2009, pp. 941–945.

[5] S. McCulloch Warren and Walter Pitts. "A logical calculus of the ideas immanent in nervous activity". In: *Bulletin of Mathematical Biophysics* 5 (1943), pp. 115–133.

[6] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. "Learning internal representations by error propagation". In: *Parallel Distributed Processing: Explorations in the Microstructure of Cognition: Foundations*. 1986, pp. 318–362.

[7] Stuart Geman, Elie Bienenstock, and René Doursat. "Neural Networks and the Bias/Variance Dilemma". In: *Neural Computing* 4.1 (1992), pp. 1–58.

[8] Andrew Y. Ng. "Feature Selection, L1 vs. L2 Regularization, and Rotational Invariance". In: *Proceedings of the Twenty-First International Conference on Machine Learning (ICML)*. 2004, p. 78.

[9] Nitish Srivastava et al. "Dropout: a simple way to prevent neural networks from overfitting". In: *Journal of Machine Learning Research* 15.1 (2014), pp. 1929–1958.

[10] Jonathan Tompson et al. “Efficient object localization using Convolutional Networks”. In: *Conference on Computer Vision and Pattern Recognition (CVPR)*. 2015, pp. 648–656.

[11] Pierre Baldi. “Autoencoders, Unsupervised Learning, and Deep Architectures”. In: *Unsupervised and Transfer Learning - Workshop held at ICML*. Vol. 27. 2012, pp. 37–50.

[12] Jacob Devlin et al. “BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding”. In: *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT*. 2019, pp. 4171–4186.

[13] Ashish Jaiswal et al. “A Survey on Contrastive Self-supervised Learning”. In: *CoRR* abs/2011.00362 (2020).

[14] Ronan Collobert and Jason Weston. “A unified architecture for natural language processing: deep neural networks with multitask learning”. In: *Proceedings of the Twenty-Fifth International Conference on Machine Learning (ICML)*. Vol. 307. 2008, pp. 160–167.

[15] Ross B. Girshick. “Fast R-CNN”. In: *International Conference on Computer Vision (ICCV)*. 2015, pp. 1440–1448.

[16] Olivier Chapelle, Bernhard Schölkopf, and Alexander Zien. “Introduction to Semi-Supervised Learning”. In: *Semi-Supervised Learning*. 2006, pp. 1–12.

[17] Ning Qian. “On the momentum term in gradient descent learning algorithms”. In: *Neural Networks* 12.1 (1999), pp. 145–151.

[18] John C. Duchi, Elad Hazan, and Yoram Singer. “Adaptive Subgradient Methods for Online Learning and Stochastic Optimization”. In: *Journal of Machine Learning Research* 12 (2011), pp. 2121–2159.

[19] Matthew D. Zeiler. “ADADELTA: An Adaptive Learning Rate Method”. In: *CoRR* abs/1212.5701 (2012).

[20] Diederik P. Kingma and Jimmy Ba. “Adam: A Method for Stochastic Optimization”. In: *3rd International Conference on Learning Representations (ICLR)*. 2015.

[21] Ilya Loshchilov and Frank Hutter. “Decoupled Weight Decay Regularization”. In: *7th International Conference on Learning Representations (ICLR)*. 2019.

[22] Timothy Dozat. “Incorporating Nesterov Momentum into Adam”. In: *ICLR Workshop*. 2016.

[23] Diederik P. Kingma and Jimmy Ba. "Adam: A Method for Stochastic Optimization". In: *3rd International Conference on Learning Representations (ICLR)*. 2015.

[24] Ilya Loshchilov and Frank Hutter. "SGDR: Stochastic Gradient Descent with Warm Restarts". In: *5th International Conference on Learning Representations (ICLR)*. 2017.

[25] Leslie N. Smith. "Cyclical Learning Rates for Training Neural Networks". In: *Winter Conference on Applications of Computer Vision (WACV)*. 2017, pp. 464–472.

[26] Leslie N. Smith and Nicholay Topin. *Super-Convergence: Very Fast Training of Neural Networks Using Large Learning Rates*. 2018. arXiv: 1708.07120.

[27] Yoshua Bengio et al. "Curriculum learning". In: *Proceedings of the 26th Annual International Conference on Machine Learning (ICML)*. Vol. 382. 2009, pp. 41–48.

[28] Pietro Morerio et al. "Curriculum Dropout". In: *International Conference on Computer Vision (ICCV)*. 2017, pp. 3564–3572.

[29] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. "On the Difficulty of Training Recurrent Neural Networks". In: *International Conference on Machine Learning (ICML)*. Vol. 28. 2013, III–1310–III–1318.

[30] Xavier Glorot and Yoshua Bengio. "Understanding the difficulty of training deep feedforward neural networks". In: *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics (AISTATS)*. Vol. 9. 2010, pp. 249–256.

[31] Kaiming He et al. "Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification". In: *International Conference on Computer Vision (ICCV)*. 2015, pp. 1026–1034.

[32] Sergey Ioffe and Christian Szegedy. "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift". In: *Proceedings of the 32nd International Conference on Machine Learning (ICML)*. Vol. 37, pp. 448–456.

[33] Shibani Santurkar et al. "How Does Batch Normalization Help Optimization?" In: *Advances in Neural Information Processing Systems 31 (NIPS)*. 2018, pp. 2488–2498.

[34] Dmitry Ulyanov, Andrea Vedaldi, and Victor S. Lempitsky. "Instance Normalization: The Missing Ingredient for Fast Stylization". In: *CoRR* abs/1607.08022 (2016).

[35] Lei Jimmy Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. "Layer Normalization". In: *CoRR* abs/1607.06450 (2016).

[36] Yuxin Wu and Kaiming He. "Group Normalization". In: *International Journal on Computer Vision* 128.3 (2020), pp. 742–755.

[37] Kaiming He et al. "Deep Residual Learning for Image Recognition". In: *Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016, pp. 770–778.

[38] Y. Lecun et al. "Gradient-based learning applied to document recognition". In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324.

[39] Yann LeCun et al. "Backpropagation Applied to Handwritten Zip Code Recognition". In: *Neural Computing* 1.4 (1989), pp. 541–551.

[40] Kunihiko Fukushima and Sei Miyake. "Neocognitron: A new algorithm for pattern recognition tolerant of deformations and shifts in position". In: *Pattern Recognition* 15.6 (1982), pp. 455–469.

[41] François Chollet. "Xception: Deep Learning with Depthwise Separable Convolutions". In: *Conference on Computer Vision and Pattern Recognition (CVPR)*. 2017, pp. 1800–1807.

[42] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. "ImageNet Classification with Deep Convolutional Neural Networks". In: *Advances in Neural Information Processing Systems 25 (NIPS)*. 2012, pp. 1106–1114.

[43] Karen Simonyan and Andrew Zisserman. "Very Deep Convolutional Networks for Large-Scale Image Recognition". In: *3rd International Conference on Learning Representations (ICLR)*. 2015.

[44] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. "U-Net: Convolutional Networks for Biomedical Image Segmentation". In: *Medical Image Computing and Computer-Assisted Intervention - MICCAI*. Vol. 9351. 2015, pp. 234–241.

[45] Kaiming He et al. "Mask R-CNN". In: *International Conference on Computer Vision (ICCV)*. 2017, pp. 2980–2988.

[46] Leonard E. Baum and John A. Eagon. "An inequality with applications to statistical estimation for probabilistic functions of Markov processes and to a model for ecology". In: *Bulletin of the American Mathematical Society* 73 (1967), pp. 360–363.

[47] Arthur P. Dempster, Nan M. Laird, and Donald B. Rubin. "Maximum likelihood from incomplete data via the EM algorithm". In: *Journal of the Royal Statistical Society: Series B* 39 (1977), pp. 1–38.

[48] A. Viterbi. "Error bounds for convolutional codes and an asymptotically optimum decoding algorithm". In: *Transactions on Information Theory* 13.2 (1967), pp. 260–269.

[49] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. "Learning representations by back-propagating errors". In: *Nature* 323 (1986), pp. 533–536.

[50] Michael I. Jordan. "Attractor Dynamics and Parallelism in a Connectionist Sequential Machine". In: *Proceedings of the Eighth Annual Conference of the Cognitive Science Society*. 1986, pp. 531–546.

[51] Jeffrey L. Elman. "Finding structure in time". In: *Cognitive Science* 14.2 (1990), pp. 179–211.

[52] Mike Schuster and Kuldip K. Paliwal. "Bidirectional recurrent neural networks". In: *Transactions on Signal Processing* 45.11 (1997), pp. 2673–2681.

[53] Alex Graves, Santiago Fernández, and Jürgen Schmidhuber. "Multi-dimensional Recurrent Neural Networks". In: *17th International Conference on Artificial Neural Networks (ICANN)*. Vol. 4668. 2007, pp. 549–558.

[54] Sepp Hochreiter and Jürgen Schmidhuber. "Long short-term memory". In: *Neural computation* 9.8 (1997), pp. 1735–1780.

[55] Kyunghyun Cho et al. "Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation". In: *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. 2014, pp. 1724–1734.

[56] Alex Graves et al. "Connectionist temporal classification: labelling unsegmented sequence data with recurrent neural networks". In: *International Conference on Machine Learning (ICML)*. Vol. 148. 2006, pp. 369–376.

[57] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. "Sequence to Sequence Learning with Neural Networks". In: *Advances in Neural Information Processing Systems 27 (NIPS)*. 2014, pp. 3104–3112.

[58] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. "Neural Machine Translation by Jointly Learning to Align and Translate". In: *3rd International Conference on Learning Representations (ICLR)*. 2015.

[59] Thang Luong, Hieu Pham, and Christopher D. Manning. "Effective Approaches to Attention-based Neural Machine Translation". In: *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. 2015, pp. 1412–1421.

[60] Colin Raffel et al. "Online and Linear-Time Attention by Enforcing Monotonic Alignments". In: *Proceedings of the 34th International Conference on Machine Learning (ICML)*. Vol. 70. 2017, pp. 2837–2846.

[61] Chung-Cheng Chiu and Colin Raffel. "Monotonic Chunkwise Attention". In: *6th International Conference on Learning Representations (ICLR)*. 2018.

[62] Jan Chorowski et al. "Attention-Based Models for Speech Recognition". In: *Advances in Neural Information Processing Systems 28 (NIPS)*. 2015, pp. 577–585.

[63] Yann N. Dauphin et al. "Language Modeling with Gated Convolutional Networks". In: *Proceedings of the 34th International Conference on Machine Learning (ICML)*. Vol. 70. 2017, pp. 933–941.

[64] Jonas Gehring et al. "Convolutional Sequence to Sequence Learning". In: *Proceedings of the 34th International Conference on Machine Learning (ICML)*. Vol. 70. 2017, pp. 1243–1252.

[65] Ashish Vaswani et al. "Attention is All you Need". In: *Advances in Neural Information Processing Systems 30 (NIPS)*. 2017, pp. 5998–6008.

[66] Zihang Dai et al. "Transformer-XL: Attentive Language Models beyond a Fixed-Length Context". In: *Proceedings of the 57th Conference of the Association for Computational Linguistics (ACL)*. 2019, pp. 2978–2988.

[67] Jiarui Fang et al. "TurboTransformers: an efficient GPU serving system for transformer models". In: *PPoPP '21: 26th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*. 2021, pp. 389–402.

[68] Iz Beltagy, Matthew E. Peters, and Arman Cohan. "Longformer: The Long-Document Transformer". In: *CoRR* abs/2004.05150 (2020). URL: `https://arxiv.org/abs/2004.05150`.

[69] Nikita Kitaev, Lukasz Kaiser, and Anselm Levskaya. "Reformer: The Efficient Transformer". In: *8th International Conference on Learning Representations (ICLR)*. 2020.

[70] Oriol Vinyals et al. "Show and tell: A neural image caption generator". In: *Conference on Computer Vision and Pattern Recognition (CVPR)*. 2015, pp. 3156–3164.

[71] Kelvin Xu et al. "Show, Attend and Tell: Neural Image Caption Generation with Visual Attention". In: *Proceedings of the 32nd International Conference on Machine Learning (ICML)*. Vol. 37. 2015, pp. 2048–2057.

[72] Sumeet S. Singh and Sergey Karayev. "Full Page Handwriting Recognition via Image to Sequence Extraction". In: *16th International Conference on Document Analysis and Recognition (ICDAR)*. Vol. 12823. 2021, pp. 55–69.

[73] Alexey Dosovitskiy et al. "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale". In: *9th International Conference on Learning Representations (ICLR)*. 2021.

[74] Xiangning Chen, Cho-Jui Hsieh, and Boqing Gong. *When Vision Transformers Outperform ResNets without Pretraining or Strong Data Augmentations*. 2021. URL: `https://arxiv.org/abs/2106.01548`.

[75] Wei Liu et al. "CPTR: Full Transformer Network for Image Captioning". In: *CoRR* abs/2101.10804 (2021). URL: `https://arxiv.org/abs/2101.10804`.

[76] Rowel Atienza. "Vision Transformer for Fast and Efficient Scene Text Recognition". In: *16th International Conference on Document Analysis and Recognition (ICDAR)*. Vol. 12821. 2021, pp. 319–334.

[77] Urs-Viktor Marti and Horst Bunke. "A Full English Sentence Database for Off-line Handwriting Recognition". In: *Fifth International Conference on Document Analysis and Recognition (ICDAR)*. 1999, pp. 705–708.

[78] S. Johansson. *Manual of Information to Accompany the Lancaster-Oslo/Bergen Corpus of British English, for Use with Digital Computers*. ICAME collection of English language corpora. 1978. ISBN: 9788299073110.

[79] Paul Voigtlaender, Patrick Doetsch, and Hermann Ney. "Handwriting Recognition with Large Multidimensional Long Short-Term Memory Recurrent Neural Networks". In: *15th International Conference on Frontiers in Handwriting Recognition (ICFHR)*. 2016, pp. 228–233.

[80] Tsung-Yi Lin et al. "Microsoft COCO: Common Objects in Context". In: *13th European Conference on Computer Vision (ECCV)*. Vol. 8693. 2014, pp. 740–755.

[81] Mark Everingham et al. "The Pascal Visual Object Classes (VOC) Challenge". In: *International Journal on Computer Vision* 88.2 (2010), pp. 303–338.

[82] Théodore Bluche. "Joint Line Segmentation and Transcription for End-to-End Handwritten Paragraph Recognition". In: *Advances in Neural Information Processing Systems 29 (NIPS)*. 2016, pp. 838–846.

[83] Théodore Bluche, Jérôme Louradour, and Ronaldo O. Messina. "Scan, Attend and Read: End-to-End Handwritten Paragraph Recognition with MDLSTM Attention".

In: *International Conference on Document Analysis and Recognition (ICDAR)*. 2017, pp. 1050–1055.

[84] Nafiz Arica and Fatos T. Yarman-Vural. "An overview of character recognition focused on off-line handwriting". In: *Transactions on Systems, Man, and Cybernetics Society* 31.2 (2001), pp. 216–233.

[85] Michael Blumenstein, Brijesh K. Verma, and H. Basli. "A Novel Feature Extraction Technique for the Recognition of Segmented Handwritten Characters". In: *7th International Conference on Document Analysis and Recognition (ICDAR)*. 2003, pp. 137–141.

[86] Manju Rani and Yogesh Kumar Meena. "An Efficient Feature Extraction Method for Handwritten Character Recognition". In: *Swarm, Evolutionary, and Memetic Computing - Second International Conference (SEMCCO)*. Vol. 7077. 2011, pp. 302–309.

[87] Amit Choudhary, Rahul Rishi, and Savita Ahlawat. "Off-line Handwritten Character Recognition Using Features Extracted from Binarization Technique". In: *AASRI Conference on Intelligent Systems and Control* 4 (2013), pp. 306–312.

[88] Amit Choudhary, Rahul Rishi, and Savita Ahlawat. "A New Character Segmentation Approach for Off-Line Cursive Handwritten Words". In: *Proceedings of the First International Conference on Information Technology and Quantitative Management*. Vol. 17. 2013, pp. 88–95.

[89] Øivind Due Trier, Anil K. Jain, and Torfinn Taxt. "Feature extraction methods for character recognition-A survey". In: *Pattern Recognition* 29.4 (1996), pp. 641–662.

[90] Guillaume Renton et al. "Fully convolutional network with dilated convolutions for handwritten text line segmentation". In: *International Journal on Document Analysis and Recognition (IJDAR)* 21.3 (2018), pp. 177–186.

[91] Ronald Rosenfeld. "Two decades of statistical language modeling: where do we go from here?" In: *Proceedings of the IEEE* 88.8 (2000), pp. 1270–1278.

[92] Wassim Swaileh. "Language Modelling for Handwriting Recognition. (Des modèles de langage pour la reconnaissance de l'écriture manuscrite)". PhD thesis. Normandy University, France, 2017.

[93] Laurence Likforman-Sulem, Abderrazak Zahour, and Bruno Taconet. "Text line segmentation of historical documents: a survey". In: *International Journal on Document Analysis and Recognition (IJDAR)* 9.2-4 (2007), pp. 123–138.

[94] Vassilis Papavassiliou et al. "Handwritten document image segmentation into text lines and words". In: *Pattern Recognition* 43.1 (2010), pp. 369–377.

[95] C. Weliwitage, A. L. Harvey, and Andrew B. Jennings. "Handwritten Document Offline Text Line Segmentation". In: *Digital Image Computing: Techniques and Applications (DICTA)*. 2005, p. 27.

[96] Markus Feldbach and Klaus D. Tönnies. "Line Detection and Segmentation in Historical Church Registers". In: *6th International Conference on Document Analysis and Recognition (ICDAR)*. 2001, pp. 743–747.

[97] Wassim Swaileh, Kamel Ait-Mohand, and Thierry Paquet. "Multi-script iterative steerable directional filtering for handwritten text line extraction". In: *5th International Workshop on Multilingual OCR (ICDAR)*. 2015, pp. 1241–1245.

[98] Sofia Ares Oliveira, Benoit Seguin, and Frédéric Kaplan. "dhSegment: A Generic Deep-Learning Approach for Document Segmentation". In: *16th International Conference on Frontiers in Handwriting Recognition (ICFHR)*. 2018, pp. 7–12.

[99] Tobias Grüning et al. "A two-stage method for text line detection in historical documents". In: *International Journal on Document Analysis and Recognition (IJDAR)* 22.3 (2019), pp. 285–302.

[100] Mélodie Boillet, Christopher Kermorvant, and Thierry Paquet. "Multiple Document Datasets Pre-training Improves Text Line Detection With Deep Neural Networks". In: *25th International Conference on Pattern Recognition (ICPR)*. 2020, pp. 2134–2141.

[101] Mélodie Boillet, Christopher Kermorvant, and Thierry Paquet. "Robust text line detection in historical documents: learning and Evaluation methods". In: *International Journal on Document Analysis and Recognition (IJDAR)* (2022).

[102] Manuel Carbonell et al. "End-to-End Handwritten Text Detection and Transcription in Full Pages". In: *Workshop on Machine Learning, WML@ICDAR*. 2019, pp. 29–34.

[103] Manuel Carbonell et al. "A neural model for text localization, transcription and named entity recognition in full pages". In: *Pattern Recognition Letters* 136 (2020), pp. 219–227.

[104] Jonathan Chung and Thomas Delteil. "A Computationally Efficient Pipeline Approach to Full Page Offline Handwritten Text Recognition". In: *Workshop on Machine Learning (WML@ICDAR)*. 2019, pp. 35–40.

[105] Bastien Moysset et al. "Learning Text-Line Localization with Shared and Local Regression Neural Networks". In: *15th International Conference on Frontiers in Handwriting Recognition (ICFHR)*. 2016, pp. 1–6.

[106] Bastien Moysset, Christopher Kermorvant, and Christian Wolf. "Full-Page Text Recognition: Learning Where to Start and When to Stop". In: *International Conference on Document Analysis and Recognition (ICDAR)*. 2017, pp. 871–876.

[107] Curtis Wigington et al. "Start, Follow, Read: End-to-End Full-Page Handwriting Recognition". In: *15th European Conference on Computer Vision (ECCV)*. Vol. 11210. 2018, pp. 372–388.

[108] Chris Tensmeyer and Curtis Wigington. "Training Full-Page Handwritten Text Recognition Models without Annotated Line Breaks". In: *International Conference on Document Analysis and Recognition (ICDAR)*. 2019, pp. 1–8.

[109] Lorenzo Quirós and Enrique Vidal. "Learning to Sort Handwritten Text Lines in Reading Order through Estimated Binary Order Relations". In: *25th International Conference on Pattern Recognition (ICPR)*. 2020, pp. 7661–7668.

[110] Lorenzo Quirós and Enrique Vidal. "Reading order detection on handwritten documents". In: *Neural Computing and Applications*. 2022, pp. 1433–3058.

[111] Thomas Plötz and Gernot A. Fink. "Markov models for offline handwriting recognition: a survey". In: *International Journal on Document Analysis and Recognition (IJDAR)* 12.4 (2009), pp. 269–298.

[112] Mounim A. El-Yacoubi et al. "An HMM-Based Approach for Off-Line Unconstrained Handwritten Word Modeling and Recognition". In: *Transactions on Pattern Analysis and Machine Intelligence (TPAMI)* 21.8 (1999), pp. 752–760.

[113] Yoshua Bengio et al. "LeRec: a NN/HMM hybrid for on-line handwriting recognition". In: *Neural Computing* 7.6 (1995), pp. 1289–1303.

[114] Michel Gilloux, Bernard Lemarié, and Manuel Leroux. "A hybrid radial basis function network/hidden Markov model handwritten word recognition system". In: *Third International Conference on Document Analysis and Recognition (ICDAR)*. 1995, pp. 394–397.

[115] Sonia Garcia-Salicetti et al. "From characters to words: dynamical segmentation and predictive neural networks". In: *International Conference on Acoustics, Speech, and Signal Processing Conference Proceedings (ICASSP)*. 1996, pp. 3442–3445.

[116] Stefan Knerr and Emmanuel Augustin. "A neural network-hidden Markov model hybrid for cursive word recognition". In: *International Conference on Pattern Recognition (ICPR)*. 1998, pp. 1518–1520.

[117] Théodore Bluche, Hermann Ney, and Christopher Kermorvant. "Tandem HMM with convolutional neural network for handwritten word recognition". In: *International*

Conference on Acoustics, Speech and Signal Processing (ICASSP). 2013, pp. 2390–2394.

[118] Andrew W. Senior and Anthony J. Robinson. "Forward-backward retraining of recurrent neural networks". In: *Advances in Neural Information Processing Systems 8 (NIPS)*. 1995, pp. 743–749.

[119] Volkmar Frinken et al. "Improved Handwriting Recognition by Combining Two Forms of Hidden Markov Models and a Recurrent Neural Network". In: *Computer Analysis of Images and Patterns (CAIP)*. Vol. 5702. 2009, pp. 189–196.

[120] Alex Graves and Jürgen Schmidhuber. "Offline Handwriting Recognition with Multidimensional Recurrent Neural Networks". In: *Advances in Neural Information Processing Systems 21 (NIPS)*. 2008, pp. 545–552.

[121] Vu Pham et al. "Dropout Improves Recurrent Neural Networks for Handwriting Recognition". In: *14th International Conference on Frontiers in Handwriting Recognition (ICFHR)*. 2014, pp. 285–290.

[122] Paul Voigtlaender, Patrick Doetsch, and Hermann Ney. "Handwriting Recognition with Large Multidimensional Long Short-Term Memory Recurrent Neural Networks". In: *International Conference on Frontiers in Handwriting Recognition (ICFHR)*. 2016, pp. 228–233.

[123] Curtis Wigington et al. "Data Augmentation for Recognition of Handwritten Words and Lines Using a CNN-LSTM Network". In: *International Conference on Document Analysis and Recognition (ICDAR)*. 2017, pp. 639–645.

[124] Joan Puigcerver. "Are Multidimensional Recurrent Layers Really Necessary for Handwritten Text Recognition?" In: *14th IAPR International Conference on Document Analysis and Recognition (ICDAR)*. 2017, pp. 67–72.

[125] Bastien Moysset and Ronaldo O. Messina. "Are 2D-LSTM really dead for offline text recognition?" In: *International Journal on Document Analysis and Recognition* 22.3 (2019), pp. 193–208.

[126] Théodore Bluche and Ronaldo O. Messina. "Gated Convolutional Recurrent Neural Networks for Multilingual Handwriting Recognition". In: *14th IAPR International Conference on Document Analysis and Recognition (ICDAR)*. 2017, pp. 646–651.

[127] R. Reeve Ingle et al. "A Scalable Handwritten Text Recognition System". In: *International Conference on Document Analysis and Recognition (ICDAR)*. 2019, pp. 17–24.

[128] Raymond W. Ptucha et al. "Intelligent character recognition using fully convolutional neural networks". In: *Pattern Recognition* 88 (2019), pp. 604–613.

[129] Mohamed Yousef, Khaled F. Hussain, and Usama S. Mohammed. "Accurate, data-efficient, unconstrained text recognition with convolutional neural networks". In: *Pattern Recognition* 108 (2020), p. 107482.

[130] Jorge Sueiras et al. "Offline continuous handwriting recognition using sequence to sequence neural networks". In: *Neurocomputing* 289 (2018), pp. 119–128.

[131] Johannes Michael et al. "Evaluating Sequence-to-Sequence Models for Handwritten Text Recognition". In: *International Conference on Document Analysis and Recognition (ICDAR)*. 2019, pp. 1286–1293.

[132] Jason Poulos and Rafael Valle. "Character-based handwritten text transcription with attention networks". In: *Neural Computing Application* 33.16 (2021), pp. 10563–10573.

[133] Denis Coquenet et al. "Have convolutions already made recurrence obsolete for unconstrained handwritten text recognition ?" In: *Workshop on Machine Learning (WML@ICDAR)*. 2019, pp. 65–70.

[134] Denis Coquenet, Clément Chatelain, and Thierry Paquet. "Recurrence-free unconstrained handwritten text recognition using gated fully convolutional network". In: *17th International Conference on Frontiers in Handwriting Recognition (ICFHR)*. 2020, pp. 19–24.

[135] Christoph Wick, Jochen Zöllner, and Tobias Grüning. "Transformer for Handwritten Text Recognition Using Bidirectional Post-decoding". In: *16th International Conference on Document Analysis and Recognition (ICDAR)*. Vol. 12823. 2021, pp. 112–126.

[136] Lei Kang et al. "Pay Attention to What You Read: Non-recurrent Handwritten Text-Line Recognition". In: *CoRR* abs/2005.13044 (2020). URL: `https://arxiv.org/abs/2005.13044`.

[137] Daniel Hernandez Diaz et al. "Rethinking Text Line Recognition Models". In: *CoRR* abs/2104.07787 (2021). arXiv: `2104.07787`. URL: `https://arxiv.org/abs/2104.07787`.

[138] Kenneth M. Sayre. "Machine recognition of handwritten words: A project report". In: *Pattern Recognition* 5.3 (1973), pp. 213–228.

[139] Philippine Barlas et al. "A Typed and Handwritten Text Block Segmentation System for Heterogeneous and Complex Documents". In: *11th International Workshop on Document Analysis Systems (DAS)*. 2014, pp. 46–50.

[140] Lorenzo Quirós. "Multi-Task Handwritten Document Layout Analysis". In: *CoRR* abs/1806.08852 (2018). URL: http://arxiv.org/abs/1806.08852.

[141] Yann Soullard et al. "Multi-scale Gated Fully Convolutional DenseNets for semantic labeling of historical newspaper images". In: *Pattern Recognition Letters* 131 (2020), pp. 435–441.

[142] Xiao Yang et al. "Learning to Extract Semantic Structure from Documents Using Multimodal Fully Convolutional Neural Networks". In: *Conference on Computer Vision and Pattern Recognition (CVPR)*. 2017, pp. 4342–4351.

[143] Martin Schall, Marc-Peter Schambach, and Matthias O. Franz. "Multi-dimensional Connectionist Classification: Reading Text in One Step". In: *13th International Workshop on Document Analysis Systems (DAS)*. 2018, pp. 405–410.

[144] Mohamed Yousef and Tom E. Bishop. "OrigamiNet: Weakly-Supervised, Segmentation-Free, One-Step, Full Page Text Recognition by learning to unfold". In: *Conference on Computer Vision and Pattern Recognition (CVPR)*. 2020, pp. 14698–14707.

[145] Denis Coquenet, Clément Chatelain, and Thierry Paquet. "SPAN: a Simple Predict & Align Network for Handwritten Paragraph Recognition". In: *International Conference on Document Analysis and Recognition (ICDAR)*. Vol. 12823. 2021, pp. 70–84.

[146] Denis Coquenet, Clément Chatelain, and Thierry Paquet. "End-to-end Handwritten Paragraph Text Recognition Using a Vertical Attention Network". In: *Transactions on Pattern Analysis and Machine Intelligence (TPAMI)* (2022).

[147] Ahmed Cheikh Rouhou et al. "Transformer-Based Approach for Joint Handwriting and Named Entity Recognition in Historical documents". In: *Pattern Recognition Letters* 155 (2022), pp. 128–134.

[148] Anoop R. Katti et al. "Chargrid: Towards Understanding 2D Documents". In: *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*. 2018, pp. 4459–4469.

[149] Christian Reisswig et al. "Chargrid-OCR: End-to-end trainable Optical Character Recognition through Semantic Segmentation and Object Detection". In: *CoRR* (2019). URL: http://arxiv.org/abs/1909.04469.

[150] Timo I. Denk and Christian Reisswig. "BERTgrid: Contextualized Embedding for 2D Document Representation and Understanding". In: *CoRR* (2019). URL: http://arxiv.org/abs/1909.04948.

[151] Yang Xu et al. "LayoutLMv2: Multi-modal Pre-training for Visually-rich Document Understanding". In: *Proceedings of the 59th Annual Meeting of the Association for*

Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (ACL/IJCNLP). 2021, pp. 2579–2591.

[152] Rafal Powalski et al. "Going Full-TILT Boogie on Document Understanding with Text-Image-Layout Transformer". In: *16th International Conference on Document Analysis and Recognition (ICDAR)*. Vol. 12822. 2021, pp. 732–747.

[153] Sanket Biswas et al. "DocSynth: A Layout Guided Approach for Controllable Document Image Synthesis". In: *16th International Conference on Document Analysis and Recognition (ICDAR)*. Vol. 12823. 2021, pp. 555–568.

[154] Olivier Galibert, Juliette Kahn, and Ilya Oparin. "The zonemap metric for page segmentation and area classification in scanned documents". In: *International Conference on Image Processing (ICIP)*. 2014, pp. 2594–2598.

[155] Denis Coquenet, Clément Chatelain, and Thierry Paquet. "DAN: a Segmentation-free Document Attention Network for Handwritten Document Recognition". In: *Under review* (2022). URL: `https://arxiv.org/abs/2203.12273`.

[156] Byungseok Roh et al. "Spatially Consistent Representation Learning". In: *Conference on Computer Vision and Pattern Recognition (CVPR)*. 2021, pp. 1144–1153.

[157] Mathilde Caron et al. "Unsupervised Learning of Visual Features by Contrasting Cluster Assignments". In: *Annual Conference on Neural Information Processing Systems (NIPS)*. 2020.

[158] Kaiming He et al. "Momentum Contrast for Unsupervised Visual Representation Learning". In: *Conference on Computer Vision and Pattern Recognition (CVPR)*. 2020, pp. 9726–9735.

[159] Ting Chen et al. "A Simple Framework for Contrastive Learning of Visual Representations". In: *International Conference on Machine Learning (ICML)*. Vol. 119. 2020, pp. 1597–1607.

[160] Aviad Aberdam et al. "Sequence-to-Sequence Contrastive Learning for Text Recognition". In: *Conference on Computer Vision and Pattern Recognition (CVPR)*. 2021, pp. 15302–15312.

[161] Mathilde Caron et al. "Emerging Properties in Self-Supervised Vision Transformers". In: *International Conference on Computer Vision (ICCV)*. 2021, pp. 9630–9640.

[162] Wenqi Zhao et al. "Handwritten Mathematical Expression Recognition with Bidirectionally Trained Transformer". In: *International Conference on Document Analysis and Recognition (ICDAR)*. Vol. 12822. 2021, pp. 570–584.