Qu'est-ce qu'est l'*Handwritten Text Recognition* ou la reconnaissance automatique d'écriture manuscrite ? L'*Handwritten Text Recognition* (HTR) est l'une des meilleurs approches aujourd'hui pour prédire du texte à partir d'une image numérique, y compris les manuscrits et les imprimés. Le développement de cette technologie occupe les chercheurs depuis plus d'un siècle. Pour expliquer comment l'HTR fonctionne, il faut comprendre dans un premier temps ce qu'est une image numérique et comment les différents éléments qui la compose : la mise en page (telle que les zones de textes et les illustrations), les lignes, ainsi que le texte proprement dit.

1 Le fonctionnement général de l'HTR

1.1 L'image numérique

L'un des objectifs de l'HTR est d'imiter l'œil humain et reconnaître les lignes et les points d'une écriture sur une image numérisée contenant du texte. Une image numérique consiste en un quadrillage qui se compose de petits carrés qui s'appellent des pixels. Issu de l'anglais, le mot pixel veut dire *picture element* et désigne l'élément minimal d'une image numérique. Les pixels sont stockés sous le format *bitmap* ou BMP et chaque pixel peut compter 1 bit, 4 bits, 8 bits, ou 24 bits selon l'encodage de l'image. Quelque soit l'encodage, chaque pixel n'a qu'une seule couleur (système RVB : rouge, vert, bleu). Vu ensemble, un groupe de pixels peut donner l'impression des courbes d'un objet ou d'un caractère. L'exemple de figure 1 montre la diagonale de la lettre « A » écrite en crayon rouge.

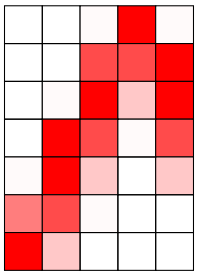


FIGURE 1 – Une couleur par pixel

Un scanner encode l'image en décomposant une image en unités de pixel et en donnant à chaque pixel un tableau de trois entiers qui conforme au système RVB (cf. Figure 2). Le premier entier qui se trouve dans la donnée tripartite d'un pixel défini l'intensité de la couleur rouge ; le deuxième entier celle de la couleur vert et le troisième celle de la couleur bleu. Avec un peu de distance, l'œil humain, à partir de ce quadrillage, distingue les formes qui constituent une image. L'HTR vise à identifier et reconnaître les points contiguës d'un caractère.

255,255,255	255,255,255	255,250,250	255,0,0	255,250,250
255,255,255	255,255,255	255,75,75	255,75,75	255,0,0
255,255,255	255,250,250	255,0,0	255,200,200	255,0,0
255,255,255	255,0,0	255,75,75	255,250,250	255,75,75
255,250,250	255,0,0	255,200,200	255,255,255	255,200,200
255,125,125	255,75,75	255,250,250	255,255,255	255,255,255
255,0,0	255,200,200	255,255,255	255,255,255	255,255,255

FIGURE 2 – Donnée RVB

1.2 Les tâches d'un logiciel HTR

Pour aboutir à une prédiction HTR, la tâche consiste en trois étapes : la reconnaissance des différentes zones de la page, la reconnaissance des lignes où se trouve du texte, et la transcription du texte L'analyse de la mise en page n'est pas obligatoire, mais les deux autres tâches sont essentielles à l'HTR. Chacune exige son propre modèle entraîné spécifiquement pour sa tâche. (cf. Figure 3)

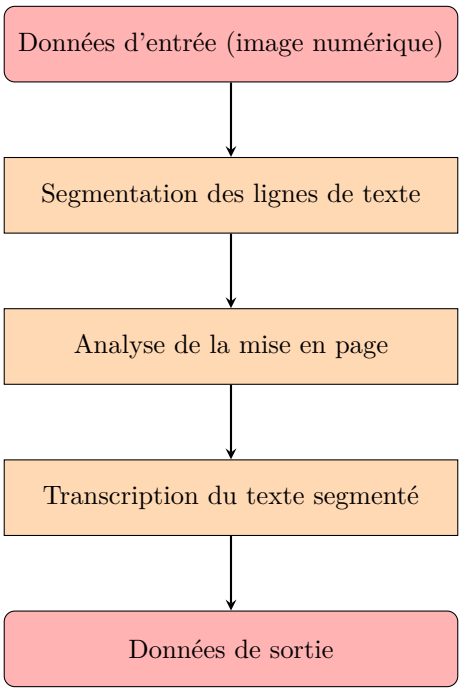


FIGURE 3 – Processus d'un logiciel HTR

1.2.1 La segmentation des zones de la page

La première tâche de la reconnaissance du texte est de localiser l'emplacement du texte. Cette étape s'appelle souvent la segmentation et elle est indispensable.[1] Selon son entraînement, un modèle de segmentation recherche les espaces entre les contours d'un caractère ou entre les lignes de texte. Si le modèle est entraîné pour reconnaître du texte écrit en japonais, par exemple, il rechercherait l'ensemble de caractères bordés à gauche et à droit de l'espace et les reconnaîtrait comme étant une ligne de texte. Ainsi que tracer la limite d'une ligne de texte, il faut aussi tracer la limite d'un caractère. L'ensemble de coordonnées qui encadre une telle entité s'appelle un « masque ».[2] Cet objet contiguë permet au modèle de reconnaître du texte. Après avoir reconnu les entités telles que les lignes de texte, contenues dans les masques, le logiciel HTR analyse ensuite la mise en page des entités, mais, comme explique Alix Chagué,

1. Alix CHAGUÉ, Thibault CLÉRICE et Laurent ROMARY. « HTR-United : Mutualisons La Vérité de Terrain ! » In : *DHNord2021 - Publier, Partager, Réutiliser Les Données de La Recherche : Les Data Papers et Leurs Enjeux*. Lille, France : MESHS, nov. 2021. URL : `https://hal.archives-ouvertes.fr/hal-03398740` (visité le 10/08/2022).

2. Denis COQUENET, Clément CHATELAIN et Thierry PAQUET. « Handwritten Text Recognition : From Isolated Text Lines to Whole Documents ». In : *ORASIS 2021*. Saint Ferréol, France : Centre National de la Recherche Scientifique [CNRS], sept. 2021. URL : `https://hal.archives-ouvertes.fr/hal-03339648` (visité le 04/08/2022).

cette étape n'est pas nécessaire à l'HTR. [3]

1.2.2 La transcription

La transcription du texte est l'étape fondamentale de l'HTR. Cette dernière s'effectue à partir des données encadrées dans les masques. Il faut ce qu'on appelle un modèle HTR [4]. La sortie d'un logiciel HTR peut être enrichie et inclure, en plus du texte brut reconnu, les données de la segmentation. En tout cas, la phase de transcription est essentiel dans un protocol de reconnaissance automatique du texte.

2 Les origines de l'HTR

L'*Handwritten Text Recognition* a évolué à partir des recherches sur l'*Optical Character Recognition*. Les origines de cette dernière méthode remontent au dix-neuvième siècle. Les progrès les plus importants de cette technologie ont eu lieu il y a plus qu'un demi-siècle aux États-Unis. Le développement des logiciels OCR a été réalisé dans le cadre de l'amélioration du traitement en masse des données numérisées. Dans les années soixante, le besoin de gérer et traiter de grandes quantités de données est devenu de plus en plus pressant notamment dans les grandes entreprises, dont les banques. La numérisation des journaux et le traitement en masse des lettres à la poste furent aussi l'un des contextes importants du développement de la reconnaissance automatique du texte sur des images.

Dans les années mille neuf cent soixante, de plus en plus de sociétés souhaitaient adopter les méthodes de traitement des données assisté par ordinateur. Ce traitement a exigé l'encodage des données dans un format directement exploitable par ordinateur. Il y avait alors deux moyens pour la saisie des données : (i) la carte perforée dans laquelle une machine faisait des trous qu'un ordinateur savait lire. En deuxième temps, (ii) la bande magnétique sur laquelle une machine imprimait des bytes, la plus petite unité exploitable par ordinateur.

L'*Optical Character Recognition* comptait sur ce dernier moyen, ainsi que sur une autre technologie, la numérisation des images par scanner. L'atout de l'OCR est qu'en principe la saisie de données se fait automatiquement. À l'époque, les autres moyens d'encoder les données textuelles exigeaient leur saisie manuelle. Dans le cas des cartes perforées, un individu lisait un document et tapait le texte à la perforatrice. Le *Magnetic Tape/Selectric Typewriter* (MT/ST), développé par l'entreprise américaine IBM en 1964, comptait également sur la saisie manuelle des données, mais les encodait sur une bande magnétique au lieu d'une carte perforée.

3. Chagué, Clérice et Romary, « HTR-United ».

4. l'acronyme HTR peut soit renvoyer uniquement à cette étape, soit plus généralement au processus complet qui comprend la reconnaîssance du texte, mais aussi la segmentation des zones et des lignes

En 1971, le chercheur Ben R. Schneider a comparé l'OCR et ces deux autres moyens d'encodage du texte. [5] Schneider a déterminé que l'OCR n'avait pas encore surpassé l'appareil MT/ST d'IBM parce que, malgré sa saisie automatique de données, il exigeait toujours beaucoup de correction à la main, car contrairement aux deux autres méthodes, les données d'entrée du traitement OCR n'étaient pas créées par un humain mais par un scanner. L'appareil MT/ST avait donc une exactitude supérieure à l'OCR à l'époque.

En outre, un logiciel OCR était limité par son jeu de données de polices, puisqu'il prédit du texte en faisant un comparaison entre un caractère qu'il a mémorisé et le motif qu'il a reconnu sur l'image. Même aujourd'hui l'OCR se distingue de son successeur l'HTR par le fait qu'il compte sur une base de données des polices. Contraire à l'OCR dans les années soixante-dix, l'appareil MT/ST pouvait encoder des documents des diverses polices en comptant sur le discernement d'un être humain lors de la saisie des données.

En 1977, Gian Piero Zarri, en résumant l'état de l'art de l'OCR, a remarqué que la reconnaissance du texte sur les manuscrits n'était pas encore faisable.

> Rappelons que la reconnaissance optique des caractères permet la lecture directe du texte par un « scanner » qui se charge d'effectuer le transfert sur bande magnétique ; le texte doit être composé avec des caractères de type « imprimerie » ou « machine à écrire », car la lecture de caractères « manuels » ne semble pas encore actuellement complètement sortie de la phase expérimentale. [6]

Dans les mêmes années, le chercheur américain Ray Kurzweil fait le même constat : *Kurzweil* ou le *Kurzweil Reading Machine* (KRM). L'OCR peut reconnaître plusieurs polices. [7] Cependant, comme dit Zarri, l'OCR reste sous la dépendance de la reconnaissance des polices et donc ne peut pas encore parvenir à la prédiction du texte écrit, ce qui a amené au développement de l'*Handwritten Text Recognition*.

2.1 La binarisation

Afin de reconnaître la police d'un caractère, les logiciels OCR du XXe siècle avaient besoin d'un processus préliminaire qui s'appelle la binarisation. Depuis quelques années, cette étape n'est plus nécessaire pour l'HTR, mais il est toujours courant pour les logiciels OCR contemporains. [8] Comme expliquent Patrick

5. Ben R. Schneider. « The Production of Machine-Readable Text : Some of the Variables ». In : *Computers and the Humanities* 6.1 (sept. 1971), p. 39-47. ISSN : 0010-4817, 1572-8412. DOI : 10.1007/BF02402324. URL : http://link.springer.com/10.1007/BF02402324 (visité le 02/08/2022).

6. Gian Piero Zarri. « Quelques aspects techniques de l'exploitation informatique des documents textuels : saisie des données et problèmes de sortie ». In : *Publications de l'École Française de Rome* 31.1 (1977), p. 399-413. URL : https://www.persee.fr/doc/efr_0000-0000_1977_act_31_1_2286 (visité le 01/08/2022).

7. Gregory Goodrich. « Kurzweil Reading Machine : A Partial Evaluation of Its Optical Character Recognition Error Rate ». In : *Journal of Visual Impairment and Blindness* (12 jan. 1979).

8. Patrick Jentsch et Stephan Porada. « From Text to Data Digitization, Text Analysis and Corpus Linguistics ». In : *Digital Methods in the Humanities : Challenges, Ideas,*

Jentsch et Stephan Porada, « The idea is to only extract the pixels which actually belong to the characters and discard any other pixel information which, for example, is part of the background [9]. » La binarisation trie les pixels d'une image en deux classes : l'arrière-plan (*background* en anglais) et le premier plan (*foreground* en anglais).

Normalement pour binariser une image, on veut remplacer la valeur composée d'un pixel, c'est-à-dire les trois degrés du rouge, du vert, et du bleu, avec la valeur simple d'un décimal. Ce décimal veut représenter l'échelle des gris dans le pixel. En anglais, ce traitement s'appelle le *grayscale* ou le niveau de gris en français. Aujourd'hui les langages de programmation ont souvent des librairies qui fournissent des méthodes pour rendre une image en niveau de gris automatiquement. Mais dans la figure 4, nous donnons un calcul simple qui permet le traitement des niveaux de gris. On commence toujours avec la donnée tripartite du pixel, qu'on veut remplacer par une seule valeur. Représentons chaque partie de la donnée du pixel par les variables p :

$$p_1, p_2, p_3$$

En premier temps, on prend la moyenne des degrés de la couleur, c'est-à-dire la moyenne de p ou $\bar{p}$. La Figure 5a montre le résultat de ce calcul superposé à l'image originale.

$$\bar{p} = \sum_{i=1}^{3} \frac{1}{3} p_i = \frac{1}{3}(p_1 + p_2 + p_3)$$

Ensuite, on récupère $\bar{p}$ et la divise par la valeur maximale du p, qui est 255 parce que le degré du rouge, vert, ou bleu d'un pixel ne monte qu'à 255. La Figure 5b montre le résultat de ce calcul.

Donnée tripartite du pixel p_1, p_2, p_3	Moyenne du pixel $\sum_{i=1}^{3} p_i$	Valeur niveau de gris du pixel $\frac{\sum_{i=1}^{3} p_i}{\max\{p_i\}}$
255,000,000	$\bar{p} = \frac{255+0+0}{3} = 85$	$\frac{\bar{p}}{255} = 0.33$
255,075,075	$\bar{p} = \frac{255+75+75}{3} = 135$	$\frac{\bar{p}}{255} = 0.53$
255,125,125	$\bar{p} = \frac{255+125+125}{3} = 168.33$	$\frac{\bar{p}}{255} = 0.66$
255,200,200	$\bar{p} = \frac{255+200+200}{3} = 218.33$	$\frac{\bar{p}}{255} = 0.86$
255,250,250	$\bar{p} = \frac{255+220+220}{3} = 251.67$	$\frac{\bar{p}}{255} = 0.99$
255,255,255	$\bar{p} = \frac{255+255+255}{3} = 255$	$\frac{\bar{p}}{255} = 1.00$

FIGURE 4 – Évaluation des pixels

Perspectives. Sous la dir. de Silke SCHWANDT. Bielefeld University Press, 2021.

9. JENTSCH et PORADA, « From Text to Data Digitization, Text Analysis and Corpus Linguistics », p. 107.

Il y a plusieurs techniques de binarisation mais toute a besoin d'un seuil (*threshold* en anglais). Le seuil nous permet à trier les pixels en les deux classes : le *background* et le *foreground*. Nos yeux font cette étape facilement, mais un ordinateur a besoin d'un algorithme. L'un des algorithmes les plus courant pour définir le seuil a été élaboré en 1979 par le chercheur Nobuyuki Otsu. [10]

La méthode Otsu de seuillage reste toujours courante dans l'HTR [11] et elle fait partie de la librairie Python `numpy`.. [12] Elle examine la variance entre les deux classes (*background* et *foreground*) pour déterminer un seuil idéal pour le jeu de données. Visualisées dans un histogramme, comme on voit dans la figure 5, les données d'un jeu de pixels devraient se lever dans deux sommets et, idéalement, une baisse profonde devrait les diviser. Cette variance veut dire qu'il y a dans l'image une distinction importante entre les contours d'un caractère et l'arrière-plan de l'image. La méthode de seuillage examine la variance entre ces deux classes, le contour et l'arrière-plan, pour générer un seuil adapté aux données.

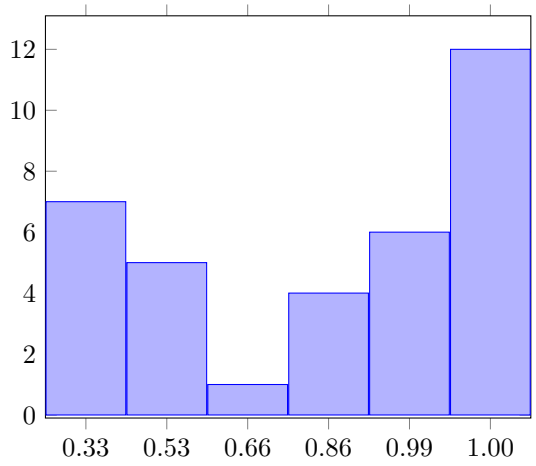


FIGURE 5 – Histogramme des valeurs niveau de gris des pixels

10. Nobuyuki OTSU. « A Threshold Selection Method from Gray-Level Histograms ». In : *IEEE Transactions on Systems, Man, and Cybernetics* 9.1 (jan. 1979), p. 62-66.

11. Imran-Ahmed SIDDIQI, Florence CLOPPET et Nicole VINCENT. « Writing Property Descriptors : A Proposal for Typological Groupings ». In : *Gazette du livre médiéval* 56.1 (2011), p. 42-57. DOI : `10.3406/galim.2011.1981`. URL : `https://www.persee.fr/doc/galim_0753-5015_2011_num_56_1_1981` (visité le 02/08/2022).

12. *Numpy : NumPy Is the Fundamental Package for Array Computing with Python.* Version 1.23.1. URL : `https://www.numpy.org` (visité le 04/08/2022).

255	255	251.67	85	251.67
255	255	135	135	85
255	251.67	85	218.33	85
255	85	135	251.67	135
251.67	85	218.33	255	218.33
168.33	135	251.67	255	255
85	218.33	255	255	255

(a) La moyenne de chaque pixel

1.00	1.00	0.98	0.33	1.98
1.00	1.00	0.53	0.53	0.33
1.00	0.98	0.33	0.86	0.33
1.00	0.33	0.53	0.98	0.53
0.98	0.33	0.86	1.00	0.86
0.66	0.53	0.98	1.00	1.00
0.33	0.86	1.00	1.00	1.00

(b) L'image en niveau de gris

0	0	0	1	0
0	0	1	1	1
0	0	1	0	1
0	1	1	0	1
0	1	0	0	0
1	1	0	0	0
1	0	0	0	0

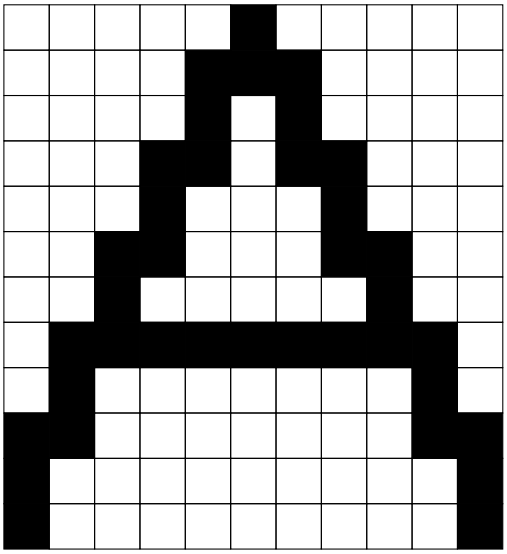
(c) L'image binarisée

FIGURE 6 – La binarisation

Le seuil sert à transformer la valeur numérique de chaque pixel soit en 0, soit en 1. Les pixels dont la valeur tombe au-dessous du seuil prennent la valeur 0 ; les autres, étant déterminés de faire partie du contour du caractère, prennent la valeur 1. La Figure 5c montre un exemple du résultat de ce triage. Ainsi, les logiciels OCR et HTR peuvent analyser les valeurs identiques et contiguës dans les données de l'image. Mais le logiciel n'est pas encore prêt à percevoir l'occurence d'un caractère.

2.2 L'algèbre linéaire, les matrices, et l'intelligence artificielle

On transforme les pixels d'un caractère en une matrice, telle que celle présentée dans la figure 6b. Pourquoi ? À la base, l'ordinateur est une calculatrice. Les logiciels OCR et HTR décomposent une image numérique en représentations mathématiques, les matrices. Ainsi, les matrices permettent aux logiciels de faire des calculs probabilistes et de prédire quel caractère correspond à quelle représentation. Mais pour faire des prédictions, un logiciel a besoin d'une intelligence artificielle.



(a) Le masque de la lettre *A*, sur l'image binarisée

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

(b) La matrice du masque de la lettre A

FIGURE 7 – La matrice d'un caractère

2.2.1 *Template matching*

Pendant la dernière moitié du XXe siècle, la reconnaissance du texte et l'intelligence artificielle ont connu des progrès importants. En 1950, il y a plus que soixante-dix ans, Alan Turing a publié sa théorisation de l'IA. [13] Pendant les années soixante, au début de l'OCR, l'IA a rendu possible la prédiction du texte représenté par les matrices en correspondant la représentation qu'un logiciel a reconnue à la représentation d'un caractère qu'il avait dans sa base de données. Cette technique primitive est le *template matching* ou le filtrage par modèle. Elle n'est pas le moyen le plus pratique. Les chercheurs V. K. Govindan et A. P. Shivaprasad l'ont apprécié en 1990, après qu'elle a été dépassée par le *feature analysis*.

> [Template matching] directly compares an input character to a standard set of prototypes stored [*modèles stockés*]. The prototype that matches most closely provides recognition. […] This type of technique suffers from sensitivity to noise and is not adaptive to differences in writing style. [14]

Sous la dépendance du *template matching*, un logiciel OCR ne parviendra pas à préduire un caractère si la totalité de sa représentation en matrice n'est pas assez proche du modèle stocké lors de sa programmation.

13. A. M. TURING. « Computing Machinery and Intelligence ». In : *Mind* LIX.236 (1er oct. 1950), p. 433-460. ISSN : 0026-4423. DOI : 10.1093/mind/LIX.236.433. URL : https://doi.org/10.1093/mind/LIX.236.433 (visité le 04/08/2022).

14. V. K. GOVINDAN et A. P. SHIVAPRASAD. « Character Recognition — A Review ». In : *Pattern Recognition* 23.7 (1990), p. 671. ISSN : 0031-3203. URL : https://www.academia.edu/6986960/Character_recognition_A_review (visité le 05/08/2022).

2.2.2 *Feature analysis*

Les progrès dans le domaine de l'IA pendant les années soixante-dix et quatre-vingt ont rendu possible le développement d'une nouvelle technique dont profite toujours les logiciels OCR, quoique avec beaucoup d'élaboration depuis. Au lieu de faire correspondre la représentation d'un caractère à une autre, les logiciels OCR décomposent un caractère en ses *features* ou ses aspects. En 1990, Govindan et Shivaprasad ont résumé l'état de cette technique.

> The features may represent global and local properties of the characters. These include strokes, and bays in various directions, end points, intersections of line segments, loops [...], and stroke relations, angular properties, sharp protrusions [...]. These features have high tolerances to distortions and style variations, and also tolerate a certain degree of translation and rotation. However, the extraction processes are in general very complex and it is difficult to generate masks for these types of features. [15]

Ainsi, le logiciel OCR essaie de faire correspondre l'ensemble de certains aspects d'une représentation à l'ensemble des mêmes aspects aux lesquels le logiciel associe un caractère. Selon Govindan et Shivaprasad, un aspect peut être un trait, le croisement des lignes, une boucle, la relation entre plusieurs traits, la caractéristique des angles, ou une saillie.

Grâce à la technique de *feature analysis*, un logiciel OCR moderne parvient à prendre une décision quant à la représentation d'un caractère bien qu'il n'ait pas trouvé le caractère de la même police, ayant la même représentation, dans sa base de données. *L'Handwritten Text Recognition* utilise aussi du *feature analysis*. Mais à la place de produire les métadonnées sur la police du caractère segmenté et reconnu ainsi que la langue du texte, l'HTR analyse les aspects d'un caractère simplement pour prédire le texte. Cependant, cette analyse n'est pas plus simple que celle réalisée par un logiciel OCR.

Le logiciel HTR a besoin d'associer beaucoup d'aspects d'une variété immense à un seul caractère puisqu'une main peut écrire un caractère par plusieurs moyens, selon la position de la lettre dans un mot. En outre, l'écriture d'une main peut varier et une page peut avoir plusieurs mains. Dit simplement, l'OCR analyse les aspects d'un caractère du point de vue d'une langue et d'une police attendue. L'HTR se focalise uniquement sur les aspects topologiques d'un caractère, sans avoir besoin de savoir la langue ni d'avoir appris la police du texte.

2.2.3 *Deep learning*

Aujourd'hui l'HTR profite du *feature analysis* ainsi que d'un développement plus récent dans l'intelligence artificielle qui s'appelle le *deep learning* ou l'apprentissage profond. Pouvant s'améliorer grâce aux réseaux neurones, un logiciel OCR ou HTR moderne peut prendre les décisions de plus en plus pertinente quand il essaie de prédire du texte à partir de l'analyse des aspects

15. Govindan et Shivaprasad, « Character Recognition — A Review ».

(*feature analysis*). L'une des premieres mises en œuvre des reseaux neurones pour *l'Handwritten Text Recognition* était le projet european *Transkribus*.[16] Un autre projet européen a suivi *Transkribus* et, contrairement au premier, laisse ses données et les architectures de ses modèles ouvertes : *Kraken*.[17] Élaboré dans l'esprit de la science ouverte, le projet *Gallic(orpor)a* a choisi d'utiliser *Kraken* et une interface de transcription, elle aussi ouverte, *eScriptorium*.

3 Le modèle HTR

Puisque *Transkribus* et *Kraken* profitent tous les deux de l'apprentissage profond, les processus mis en œuvre par les interfaces graphiques *Transkribus* et *eScriptorium* ressemblent généralement à la même description. Ils commencent avec l'entraînement d'un modèle HTR. Ensuite, ils appliquent le modèle entraîné aux données d'entrées, c'est-à-dire l'image d'un page numérisée. Le taux de réussite se calcule par le pourcentage des caractères que le modèle n'a jamais vus mais qui étaient bien prédits.

3.1 Les données d'entrée

En premier temps, avant de penser à l'entraînement d'un modèle, il faut bien connaître sa saisie des données. Les données d'entrée d'un modèle vont être les images numériques, composées des pixels. En général, leurs contenus textuels devraient se ressembler afin que le modèle se spécialise dans une écriture particulière. Il est pourtant possible d'entraîner un modèle très généraliste, mais il nécessite un très large corpus d'entraînement.

3.2 Les données d'entraînement

Le premier défi de l'entraînement d'un modèle HTR est la création des données d'entraînement. Ces données sont des transcriptions annotées et corrigées à la main à partir des images. La paire formée par une image et sa transcription s'appelle une vérité de terrain ou *ground truth* en anglais, puisque la transcription doit être parfaite ou *vraie*[18]. Afin d'entraîner un modèle HTR, il faut un jeu des vérités de terrain. Alix Chagué décrit les vérités de terrain comme,

16. Guenter Muehlberger et al. « Transforming Scholarship in the Archives through Handwritten Text Recognition : Transkribus as a Case Study ». In : *Journal of Documentation* 75.5 (9 sept. 2019), p. 954-976. issn : 0022-0418. doi : 10.1108/JD-07-2018-0114. url : https://www.emerald.com/insight/content/doi/10.1108/JD-07-2018-0114/full/html (visité le 04/08/2022).

17. Benjamin Kiessling. « Kraken - an Universal Text Recognizer for the Humanities ». In : ADHO 2019 - Utrecht. 2019. url : https://dh-abstracts.library.cmu.edu/works/9912 (visité le 10/08/2022).

18. David Lassner et al. « Publishing an OCR Ground Truth Data Set for Reuse in an Unclear Copyright Setting. Two Case Studies with Legal and Technical Solutions to Enable a Collective OCR Ground Truth Data Set Effort ». Version 1.0. In : *Fabrikation von Erkenntnis – Experimente in den Digital Humanities. Hg. von Manuel Burghardt* Lisa Dieckmann (2021). Avec la coll. d'Herzog August Bibliothek, 5). doi : 10.17175/SB005_006. url : https://zfdg.de/sb005_006 (visité le 10/08/2022).

> des ensembles de données annotées et corrigées de manière à fournir au modèle des paires composées d'une part d'une image ou d'une portion d'image (entrée) et d'autre part de l'annotation attendue (sortie), qui peut être des coordonnées dans le cas de la segmentation ou un ensemble de caractères pour la transcription. [19]

Lors de l'apprentissage, les vérités de terrain fournissent au modèle en cours son résultat souhaité pour qu'il puisse savoir comment s'améliorer afin de produire les bonnes prédictions ou les prédictions *vraies*. Les données produites reproduire au plus près la prédiction idéale des vérités de terrain. Grâce à l'apprentissage profonde, un modèle peut apprendre comment arriver à la prédiction souhaitée à partir de données d'entraînement.

3.3 L'entraînement

Lors de l'entraînement, le modèle HTR (comme un modèle TAL)) s'évalue périodiquement et une dernière fois quand l'entraînement est terminié. La dernière évaluation s'appelle le *score*. Afin d'obtenir un meilleur *score*, on peut optimiser l'entraînement du modèle en appuyant sur plusieurs paramètres.

Par exemple, on peut modifier le nombre de fois que le modèle révise sa manière de faire ses tâches de prédiction. Chaque essaie s'appelle un époch, dérivé de l'anglais *epoch*, et un plus grand nombre d'époch donne au modèle plus d'essais à s'améliorer. Cependant, plus d'époch pèse plus sur le budget d'un projet puisqu'il consomme plus de puissance de calcul et plus de temps. En outre, on ne veut pas faire passer trop d'époch et risquer le sur-apprentissage d'un modèle, qui s'appelle le *overfitting* en anglais. Cela veut dire que la fonction prédictive du modèle s'est trop bien adaptée à ses données d'entraînement, y compris tout le bruit des données, et elle n'est pas suffisamment généraliste pour réussir sur des données que le modèle n'avais jamais vues. On peut également régler la taille du pas d'apprentissage après chaque époch, c'est à dire son *learning rate*.

On peut aussi modifier la composition du jeu de données traitée lors d'un époch. Ce dernier paramètre s'appelle un *batch size*, et il est aussi un entier, comme l'époch. Disons qu'on veut entraîner notre modèle sur 400 images. Un époch prendrait trop de temps et trop de puissance de calcul s'il essayait de traiter toutes les images de notre jeu de données au même temps. Du coup, on veut le diviser selon notre paramètre *batch size*. Si on déclarait un *batch size* de 100 images, on dirait au modèle qu'il faut itérer sur son jeu de données 4 fois afin de compléter un époch, en rappelant qu'un époch est égal à une passe complète sur le jeu de données d'entraînement, voir la figure 8. Ce qu'on appelle l'erreur, soit la différence entre la prédiction du modèle et la vérité de terrain, se calcule à chaque itération d'un *batch* dans un époch. On veut que cette valeur diminue, c'est à dire que la prédiction du modèle se rapproche de plus en plus à la vérité de terrain. Pour optimiser le modèle, on peut choisir entre plusieurs fonctions pour calculer l'erreur, ce qu'on appelle une *loss function*. Dans l'exemple de la Figure 8, on pourrait appliquer une *loss function*, telle que le *mean squared error*

19. Chagué, Clérice et Romary, « HTR-United ».

(MSE), aux toutes les prédiction d'un *batch* afin de connaître l'érreur ou le *loss* du modèle à ce point de l'entraînement.

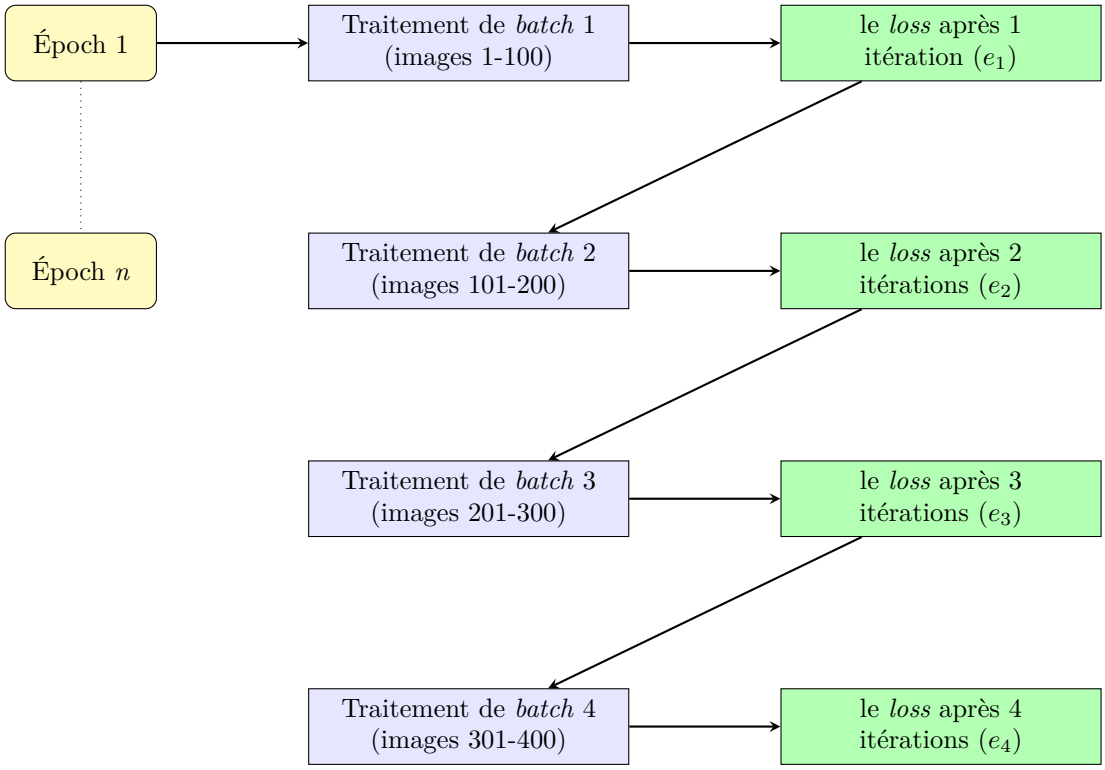


FIGURE 8 – La visualisation d'époch 1

Dans le cadre du projet *Gallic(orpor)a*, Ariane Pinche a entraîné des modèles HTR sur le corpus gold que l'équipe a fait à la main et qu'elle et Simon Gabay ont vérifié. Pinche l'a scindé en trois. Une majorité des images (80%) compose le jeu d'entraînement (*training set*). Une petite partie (10%) compose le jeu de validation (*development set*). Et la dernière partie (10%) compose le jeu de test (*testing set*).

FIGURE 9 – La division du corpus gold pour l'entraînement d'un modèle

3.4 Le résultat de l'entraînement

Le résultat de l'entraînement s'appelle un modèle. Étant spécialisé dans une écriture particulière, le modèle peut prédire du texte sur les images qui res-

semblent aux données d'entraînement. Les chercheurs s'occupent du développment des nouveaux modèles HTR qui se spécialisent dans certains manuscrits et certains imprimés historiques. Au contraire, le projet *Gallic(orpor)a* vise à entraîner plusieurs modèles. Un se spécialisera dans l'écriture des manuscrits et des incunables. Un autre modèle s'entraînera pour les imprimés.

Cependant, tous les modèles du projet *Gallic(orpor)a* se spécialisent dans les écritures latines. En théorie, un modèle HTR n'est pas limité aux polices et aux langues. Mais, jusqu'au présent, tout modèle efficace compte sur une écriture spécifique. Donc, un modèle entraîné sur les imprimés en français du XVIIe siècle peut cependant produire des bonnes prédictions pour une imprimée du même siècle en italien. Par contre, sa prédiction d'un texte écrit en arabe ne parviendra pas au même taux de réussite que le modèle atteindrait pour une imprimée en français. Le modèle ne saurait pas segmenter l'écriture arabe puisqu'il s'est appris comment reconnaître les caractères en cherchant les espaces attendues dans une écriture latine.