Institut National des Langues et Civilisations Orientales

Département Textes, Informatique, Multilinguisme

Reconnaissance optique de sinogrammes

Cas d'usage du dictionnaire mandarin-taiwanais de Wu Shou-li/Ngôo Siù-lè

MASTER

TRAITEMENT AUTOMATIQUE DES LANGUES

Parcours :

Ingénierie Multilingue

par

Afala PHAXAY

Directeur de mémoire :

Pierre Magistry

Année universitaire 2021/2022

Remerciements

Je tiens tout d'abord à remercier mon directeur de mémoire Pierre Magistry, pour sa patience, ses suggestions et ses conseils sans jamais me lâcher. Un grand merci à Damien Nouvel dont j'ai pu suivre les enseignements en Licence et en Master et aux autres professeurs de l'INALCO. Mention spécial à Marie-Anne Moreaux qui m'a accepté en Licence de traitement numérique multilingue débutant mon parcours en traitement automatique des langues. Toute ma gratitude à l'équipe Ertim qui m'a chaleureusement accueilli, qui m'ont fait me sentir intégré et avec qui j'ai partagé d'agréable repas de midi. Je suis reconnaissante à toutes les personnes que j'ai pu rencontrer et connaître à travers mon stage et avec qui j'ai pu créer un lien d'amitié pour certains. Je remercie Lili Wu, Maeva Leproux et mes camarades pour leur compagnie durant les deux années de Master.

Enfin, j'exprime toute ma reconnaissance envers ma famille, ma mère, mon père et mon frère qui m'ont toujours soutenu durant mes années d'étude.

Résumé

Dans l'ère du numérique, de nombreux supports physiques deviennent encombrants et ne permettent plus une diffusion des informations qu'ils contiennent. C'est le cas des dictionnaires. Quelque soit leur taille, les dictionnaires physiques ne sont pas des outils simple d'utilisation pour toutes personnes qui souhaitent utiliser à l'extérieur de chez soi. Faire une recherche dans un dictionnaire électronique en ligne ou hors-ligne avec son téléphone devient un automatisme pour les utilisateurs qui délaissent les versions imprimées pour une interface plus conviviale et un accès simplifié. Et bien que les dictionnaires soient maintenant plus accessibles, ce sont surtout les dictionnaires récents qui sont concernés. Si aujourd'hui les dictionnaires sont conçus directement avec des outils numériques, certains ouvrages de référence plus anciens ne sont disponibles qu'en version imprimée et il peut être intéressant de les numériser. Dans un effort de préservation et d'une meilleure diffusion, on utilise la technique de reconnaissance optique de caractères pour reconstituer le contenu de ses ouvrages.

Ce mémoire retrace le travail effectué pour la récupération des informations manquantes du dictionnaire mandarin-taiwanais de Wu Shou-li/Ngôo Siù-lè à l'aide de la reconnaissance optique de caractères et avec comme outils eScriptorium et kraken.

Mots clés : ROC, reconnaissance optique de caractères, OCR, eScriptorium, kraken, open source, numérisation de dictionnaire, mandarin-taïwanais, bopomofo, *zhuyin fuhao* (注音符號)

Table des matières

LISTE DES FIGURES

LISTE DES TABLEAUX

INTRODUCTION

Présentation générale

Dans le cadre du master Traitement Automatique des Langues (TAL) spécialité ingénierie multilingue de l'Inalco, le stage entrepris au sein de l'équipe Ertim a donné lieu à ce mémoire. L'objectif est de pouvoir reconstituer l'intégralité du dictionnaire 《國臺對照活用辭典》 de Ngôo Siú-lé (吳守禮) dans un format exploitable et ce par la reconnaissance optique de caractères (ROC ou OCR) sur les pages manquantes à l'aide de celles existantes. Les résultats d'une océrisation sont rarement parfaites et ils contiendront sûrement des erreurs qui nécessiteront que l'on repasse derrière. Mais dans l'objectif d'obtenir des résultats demandant le moins de corrections manuelles, nous allons entraîner un modèle spécifique pour notre dictionnaire.

Dans ce but, nous utilisons un système de ROC déjà existant ayant une interface graphique à disposition en open source : eScriptorium-kraken.

Problématique

L'objectif de ce mémoire est d'apporter une réflexion sur les questions suivantes :

— quels sont les étapes d'un projet de reconnaissance de caractères ?
— comment utiliser les données ?
— comment évaluer ?

A travers notre projet d'océrisation, nous souhaitons répondre à ces questions.

Plan de lecture

Nous commencerons le mémoire par le chapitre 1 dans lequel nous aborderons la numérisation et comment le projet s'inscrit-il dedans. Dans le chapitre 2, nous ferons un état de l'art en retraçant l'histoire de la reconnaissance optique des caractères de manières générales puis nous verrons brièvement la reconnaissance optique des sinogrammes avant de poursuive avec les mesures d'évaluations disponibles pour les systèmes d'OCR. Nous continuerons dans le chapitre 3 par la descriptions des outils utilisés. Le chapitre chapitre 4 sera consacré aux caractéristiques des données disponibles, des pré-traitements appliqués, du processus de génération des données artificielles et des corpus générés pour l'entraînement des modèles. Le chapitre chapitre 5 réunira les résultats obtenus sur chacune des expériences. Nous discuterons par la suite dans le chapitre 6 des travaux effectués avec les outils utilisés et des possibilités d'améliorations dans de futurs travaux. Enfin, nous terminerons par une conclusion globale.

Première partie

Contexte général

CHAPITRE 1

PRÉSENTATION DU PROJET

Sommaire

1.1 Introduction

Dans ce chapitre, nous présentons le contexte de notre travail. Dans un premier temps, nous introduirons le concept de numérisation. Ensuite, nous ferons le lien avec le format de notre objet qui est le dictionnaire. En dernier, nous présenterons l'histoire autour de ce dictionnaire sur lequel nous travaillons.

1.2 Pourquoi numériser ?

> “La numérisation est un procédé permettant de représenter des objets (image, son, signal) par des suites de nombres binaires (0 et 1).”
>
> *Le Larousse*

Cette représentation des objets permet de réduire non seulement la place que prennent ces informations physiquement, mais a aussi simplifié les échanges et leur accès au plus grand nombre. En effet, avant internet et les réseaux informatiques, la récupération et l'utilisation des documents nécessitaient du temps, de l'argent et de la main-d'œuvre. Un fichier numérique présente de nombreux avantages vis-à-vis d'une version physique. Les données prennent moins de place : des millions de documents peuvent être stockés sur un serveur. Il devient beaucoup plus simple de le consulter, de l'envoyer ou de le partager, de le modifier, de le supprimer ou de juste l'archiver. Il devient également possible d'effectuer les traitements que peut subir une donnée informatique selon le format dans lequel est le fichier. Et, même si nous sommes actuellement dans l'ère de la dématérialisation, le phénomène de la numérisation et de la dématérialisation n'est pas récent[Terras, 2011]. Les domaines de l'information, de la culture et des héritages ont rapidement embrassé les nouvelles techniques de numérisation dès leurs disponibilités.

Dans ces domaines, la numérisation a grandement contribué à la préservation et à l'accessibilité d'héritage culturel. Les ressources qu'elle offre à la population sont nombreuses. Ces ressources peuvent servir à l'apprentissage, à l'enseignement, à la recherche, à la documentation et à l'information. [Deegan and Tanner, 2004] résume les avantages de la numérisation en quelques points :

— l'accès immédiat aux objets souvent utilisés
— un accès plus simple à certaines informations contenues dans des objets spécifiques (des articles dans un journal, des entrées de dictionnaires)
— la possibilité de remettre en circulation des ouvrages en rupture de stock (Les ouvrages du XXème siècle sont toujours soumis et protégés par les droits d'auteurs. Mais ces ouvrages sont difficilement accessibles au public, car indisponible. Si elles avaient une version dématérialisée, ce ne serait plus un problème. De plus, ces ressources numérisées ont plus de chance de ne pas disparaître en ligne et deviendront un héritage.)
— l'accès à des objets dont les formats rendent leur utilisation compliquée (les cartes, les gros volumes, etc...)
— permettre une réunion virtuel de certaines collections dispersées dans le monde
— la possibilité de modifier des images pour améliorer leur clarté
— la possibilité de conserver les objets fragiles tout en présentant une copie dans différentes formes
— créer des liens entre les contenus (les références dans les textes, les vidéos d'illustration, etc...)
— la recherche dans un texte

En France, sous la législation, les responsabilités de la préservation par numérisation se divisent au sein des institutions suivantes[Beagrie, 2003] :

— la Mission des Archives nationales
— la Bibliothèque nationale de France pour les documents, les vidéos et les œuvres multimédias
— le Centre National de la Cinématographie pour les films
— l'Institut national de l'Audiovisuel pour émissions de radio et de télévision

Même si la numérisation et les services numériques qui en découlent ont parfois remplacé des produits. Elle a aussi permis l'émergence de nouveaux produits qui sont venus enrichir les produits déjà existants. Le domaine qui a su le mieux s'adapter est celui de la musique. Avec internet, il est vrai qu'à un moment elle a rencontré des difficultés avec la vente physique, mais elle s'en est remise en offrant elle-même une disponibilité digitale de leur contenu.

Même l'écriture de ce mémoire aurait été difficile sans la numérisation et l'archivage des données. Toutes les informations contenues dans les articles scientifiques, ouvrages et magazines ne se trouvant pas dans un lieu précis accessible à tous.

1.3 Numériser les dictionnaires

Qu'est-ce qu'un dictionnaire ? Quelle est l'origine de ce mot et que signifie-t-il ?

Le mot *dictionnaire* apparaît vers 1501 et provient du latin médiéval *dictionarius* de 1220 qui dérive du latin classique *dictio* signifiant "action de dire, propos, mode d'expression". C'est un ouvrage qui recueille des mots et des expressions. Le lexicographe Antoine Furetière établit une définition du terme *dictionnaire* dans son Dictionnaire universel sortie en 1690 par "Recueil fait en manière de catalogue de

tous les mots d'une langue, ou d'une ou plusieurs sciences.”. Mais cette définition ne nous renseigne pas énormément sur l'intérêt qu'il y a à la compilation des mots d'une langue.

La définition dans le Trésor de la Langue Française est la suivante “Recueil de mots d'une langue ou d'un domaine de l'activité humaine, réunis selon une nomenclature d'importance variable et présentée généralement par ordre alphabétique, fournissant sur chaque mot un certain nombre d'informations relatives à son sens et son emploi et destiné à un public défini.”

“Pour moi, le dictionnaire n'est pas un répertoire de mots, mais une représentation de la société qui change constamment. Un dictionnaire n'est pas une Bible et il faut savoir le désacraliser.” par Anaïs Vaquette, lexicographe. Elle explique bien qu'à travers différentes définitions de périodes distinctes et de type de dictionnaire, nous pouvons voir des changements dans la définition du même terme. Les dictionnaires d'époques différentes sont donc d'importantes ressources sur les langues qui peuvent nous donner et montrer les changements de la société. Durant la pandémie, l'Académie française préconise le féminin pour le genre de *“la covid 19”* alors que le dictionnaire le Robert définit *covid 19* comme pouvant être un nom masculin ou féminin.

Il y a bien sûr un aspect pratique à la numérisation du dictionnaire. La version papier doit être rééditée pour les corrections de fautes ou de coquille, les ajouts de nouveaux mots ou les redéfinitions peuvent survenir à n'importe quel moment. Pour cet aspect, la version en ligne de dictionnaire permet plus de souplesse. De plus, pour les utilisateurs, un dictionnaire électronique est contenu dans un portable permettant l'utilisation n'importe où et n'importe quand tant que l'accès à internet est disponible.

Nous avons un exemple d'un dictionnaire informatisé en France, la numérisation du Trésor de la Langue Française[1] Le Trésor de la Langue Française Informatisée donne trois niveaux de consultation possible[Pierrel et al., 2004] avec des options[Bernard, 2010] :

- Consultation simple des articles avec un correcteur d'erreur au cas où l'on n'est pas très bon en orthographe ou l'on ne connaît pas du tout le mot, avec l'utilisation de listes défilantes (n'affichent que les entrées principales) ou avec la saisie phonétique qui va donc de pair avec le correcteur
- La recherche assistée en remplissant un formulaire avec 5 possibilités d'affiner la recherche
- La recherche complexe permet d'imposer la place du mot qu'on recherche dans les résultats ou de faire une recherche .

[Barque et al., 2010] propose une utilisation différente d'un simple dictionnaire en transformant le TLFI en une base de données de la langue française.

1.4 Le cas du “guo tai dui zhao huo yong ci dian” 《國臺對照活用辭典》

Le dictionnaire 《國臺對照活用辭典》 est l'œuvre de toute une vie de son auteur : professeur 吳守禮 Ngôo Siú-lé, décédé en 2005 alors âgé de 97 ans. Linguiste dont les recherches se portaient principalement sur le mandarin et le taiwanais. Il a toujours eu envie de compléter les dictionnaires mandarins existants. Après la seconde

1. http://atilf.atilf.fr/

guerre mondiale, dans son enthousiasme de la pratique du mandarin, il s'est profondément rendu compte du manque de dictionnaires chinois et lorsqu'il a retrouvé une note de 1950 sur laquelle il a écrit "vocabulaires non trouvés dans le dictionnaire chinois", il a décidé d'entreprendre son projet. Durant les années qui suivent, son projet avancera à petits pas avec des pauses suite à des changements dans sa vie personnelle et professionnelle. Il y aura finalement consacré au moins 20 ans pour compiler sa connaissance de la langue et produire son dictionnaire. Bien que paru il y a une de cela une vingtaine d'année en 2000, son travail a commencé il y a longtemps. Le contenu en taiwanais a posé énormément de problèmes à l'auteur, car il existe une prononciation, mais pas de caractère officiellement reconnu comme "le caractère" pour l'illustrer. Dans sa préface, l'auteur souligne le fait qu'étant donné le temps que l'ouvrage a pris pour sortir, car il est difficile de trouver les bonnes correspondances au taiwanais. Le besoin et l'accès aux informations contenues dans cet ouvrage sont restreints. Son contenu n'est sans doute plus vraiment d'actualité, car depuis qu'il a commencé, d'autres dictionnaires sont sortis. Son dictionnaire est le résultat de ses connaissances et de son expérience qui datent d'une autre période. Son accent et son vocabulaire sont ancrés dans son temps alors que la langue subit des changements rendant son dictionnaire "obsolète".

La numérisation du dictionnaire est avant tout une demande de la famille du défunt pour poursuivre son souhait de promouvoir et de diffuser le taiwanais [台灣維基媒體協會 , 2018]. Les héritiers légitimes signent en 2015 un contrat de licence avec Wikimedia Taiwan. La famille donne le droit de numériser et d'informatiser entièrement le dictionnaire et leur autorise la publication, le partage et la distribution en ligne sous la licence CC-BY-SA. En réalité, l'auteur, bien qu'âgé, était bien versé dans l'informatique puisqu'en effet, il commença à éditer son dictionnaire sur ordinateur à partir de 1993 et garda une copie du contenu sur des disquettes. Cependant, les disquettes étaient encore présentes, mais commençaient déjà à être remplacé à la fin des années 1990 par d'autres supports tel que le disque compact et internet puis la clé usb et les cartes mémoires dans les années 2000. Les disquettes ont survécu au temps mais ne sont pas toutes restées indemnes. Les données d'une des disquettes étaient corrompues.

Les informations déjà récupérées sont disponibles sur :

— son site dédié `https://koktai.github.io/`

— `https://zh.m.wikisource.org/wiki/%E5%9C%8B%E8%87%BA%E5%B0%8D%E7%85%A7%E6%B4%BB%E7%94%A8%E8%BE%AD%E5%85%B8`

1.4.1 Le Taiwanais

Bien que le mandarin soit la langue officielle de Taiwan depuis la fin de l'occupation japonaise en 1945, le taiwanais est parlé par près de 70% des habitants de l'île. Il fait parti de la famille des langues sino-tibétaine et est issu du minnan qui est parlé par des millions de personnes dans le monde et principalement par les habitants du sud de la Chine. [Klöter, 2005, Heylen and Kempf, 2001]

Le système du *zhuyin fuhao* (注音符號) ou *bopomofo* (ㄅㄆㄇㄈ)

Le système du *zhuyin fuhao* (注音符號 ou *bopomofo* en Occident qui traduit littéralement signifie "symboles phonétiques" a été créé par la Commission sur l'unification de la prononciation de 1912-1913 sous La République de Chine pour devenir l'outil le plus adéquat à l'apprentissage du mandarin. Il est étendu pour couvrir les

sons du taïwanais et est utilisé tout au long du dictionnaire. L'appellation *bopomofo* est l'équivalent des termes *azerty* ou *qwerty* pour désigner l'arrangement des claviers et correspond aux quatre premiers symboles :

— *bo* (ㄅ) pour le son b de "papa" /p/
— *po* (ㄆ) pour le son p expiré /p^h/
— *mo* (ㄇ) pour le son m de "maman" /m/
— *fo* (ㄈ) pour le son f de "famille" /f/

Le système comporte actuellement 24 consonnes et 16 voyelles, ainsi que 4 tons.

zhùyīn	pīnyīn	API		zhùyīn	pīnyīn	API		zhùyīn	pīnyīn	API		zhùyīn	pīnyīn	API
Consonnes (ou 聲母 shēngmǔ)														
ㄅ	*b*	b̥		ㄆ	*p*	p^h		ㄇ	*m*	m		ㄈ	*f*	f
ㄉ	*d*	d̥		ㄊ	*t*	t^h		ㄋ	*n*	n		ㄌ	*l*	l
ㄍ	*g*	g̊		ㄎ	*k*	k^h		ㄏ	*h*	x				
ㄐ	*j*	d͡ʑ̥		ㄑ	*q*	$t\!\!\not\!\!c^h$		ㄒ	*x*	ɕ		ㄧ	*y*	j
ㄓ	*zh*	d͡ʐ̥		ㄔ	*ch*	$t\!\!\not\!\!s^h$		ㄕ	*sh*	ʂ		ㄖ	*r*	ʐ
ㄗ	*z*	d͡z̥		ㄘ	*c*	ts^h		ㄙ	*s*	s		ㄨ	*w*	w
Voyelles (ou 韻母 yùnmǔ)														
ㄚ	a	a		ㄛ	*o*	o		ㄜ	*e*	ɤ		ㄝ	*ê*	ɛ
ㄞ	ai	a^i		ㄟ	*ei*	e^i		ㄠ	*ao*	$a^ʊ$		ㄡ	*ou*	o^u
ㄢ	an	an		ㄣ	*en*	ən		ㄤ	*ang*	ɑŋ		ㄥ	*eng*	əŋ
ㄩ	*ü*	y		ㄧ	*i*	i		ㄨ	*u*	u		ㄦ	er	ə˞
Tons (ou 聲調 shēngdiào)														
1 : -	¯			2 : ˊ	ˊ			3 : ˇ	ˇ			4 : ˋ	ˋ	

FIGURE 1.1 – Système *zhuyin fuhao* (注音符號) / Bopomofo
ref, https://fr.wikipedia.org/wiki/Bopomofo

1.5 Conclusion

Nous avons pu voir que la numérisation est une action bénéfique dans des domaines très variables dont on reçoit les bénéfices au quotidien. Numériser les dictionnaires n'est pas une perte de temps que ce soit d'un point de vue individuel ou collectif.

CHAPITRE 2

ÉTAT DE L'ART

Sommaire

2.1 Introduction

La reconnaissance optique de caractère est un champ de recherche pluridisciplinaire qui touche à l'intelligence artificielle (AI), le traitement automatique des langues (NLP) et la vision par ordinateur (computer vision). Nous utilisons ces techniques au quotidien sans y prêter attention alors qu'elles nous entourent.

Dans ce chapitre, nous présentons une brève chronologie de la reconnaissance automatique de caractères. Puis nous verrons les différentes techniques utilisées pour atteindre cet objectif. On poursuivra avec les étapes de l'océrisation avant de continuer sur ce qui s'est fait sur la reconnaissance automatique de caractères autour du chinois. Enfin, nous terminerons sur les mesures dévaluation.

2.2 Types de ROC

Quand on parle de la reconnaissance optique de caractères, on n'identifie pas qu'un seul type de caractères. La reconnaissance optique de caractères s'occupe de reconnaître toutes formes de caractères traités. Lorsque le processus d'océrisation se déroule sur un texte déjà écrit, que ce soit manuscrit ou imprimé, alors on appelle cela de la reconnaissance optique de caractères *off-line*. Son contraire est *on-line* qui est la reconnaissance des caractères alors qu'ils sont en train d'être écrits. Ce qui ne peut s'appliquer qu'à l'écriture manuscrite.

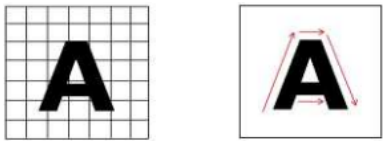


FIGURE 2.1 – Reconnaissance off-line (gauche), reconnaissance on-line (droite) ref, [Rao et al., 2016]

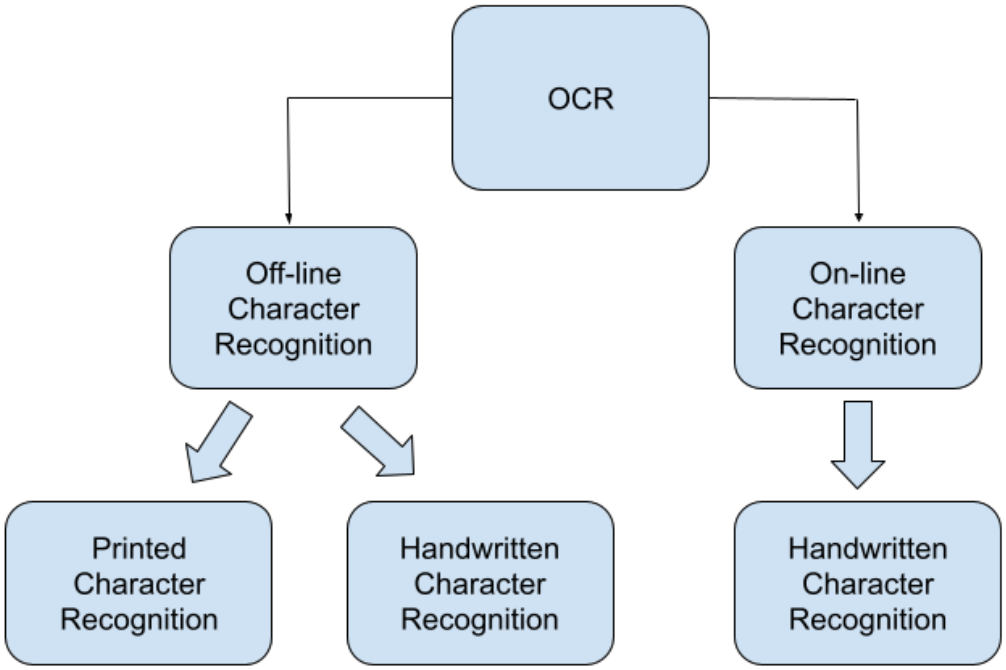


FIGURE 2.2 – Types de reconnaissance optique de caractères

2.3 ROC, d'hier à aujourd'hui

La reconnaissance optique de caractères ne date pas d'aujourd'hui. Le besoin de reproduire l'action humaine d'enregistrer visuellement ce qu'on lit puis d'envoyer les informations reconnues au cerveau pour qu'il puisse reconstituer l'image date de la fin du XIXème siècle. Le précurseur date de 1870 avec le scanner à rétine de Charles R. Carey qui est considéré comme la première invention d'un système OCR. Durant les années qui suivent, plusieurs tentatives de création d'un outil d'OCR pour aider les malvoyants à lire des documents seront mises en oeuvre. [Mori et al., 1992]

Dépôt de brevet d'un système OCR en 1929 en Allemagne par Gustav Tauschek puis en aux Etats Unis en 1935, il invente une machine qui compare le résultat d'un scan à un des modèles mémorisés. Cependant ce ne seront qu'à partir du milieu des années 1940 avec les avancées technologiques qu'apparaissent les OCR que l'on connaît maintenant. À partir des années 1960 il y a trois génération d'OCR [Patil et al., 2015] :

1ère génération

Ce qu'on peut considérer comme les premiers systèmes d'OCR à but commerciale sont apparus durant les années 1960 et 1965. Ce qui distinguera cette génération

d'OCR, c'est la lecture restreinte de caractères. Ces symboles étaient spécialement conçus pour être lu par des machines et ne semblaient alors pas naturels. Avec la création de motif de caractères, des machines pour des typographies multiples sont apparues. Elles pouvaient inclure jusqu'à 10 polices différentes. Le nombre de police qu'elles pouvaient reconnaître étant bien sûr limité par la méthode de reconnaissance de motifs qui était le *template matching* et qui consistait à comparer l'image du caractère à une bibliothèque de prototypes d'image pour chaque caractère et chaque police.

2ème génération

Entre la période des années 60 et début 70, sont apparues les machines de seconde génération. Elles pouvaient reconnaître les caractères imprimés standard et commençaient à reconnaître les caractères imprimés à la main, reconnaissant les caractères numériques et quelques lettres et symboles. Le premier de ce genre a été le IBM 128. Dans cette même période, Toshiba révolutionne avec la première machine à trier pour les codes postaux et Hitachi développe la première machine qui permet de bonne performance à bas coût. En 1966, des ingénieurs de l'ECMA ont créé la police de caractères OCR-A[PlanèteTypographie, nd] qui est un set de caractères optimisé pour la reconnaissance optique mais un peu plus ardu pour l'oeil humain. Elle répondait aux critères élaborés par l'*US Bureau of Standards*, Ce set se conforme aux normes de 1981 de l'*American National Stadards Institute* (ANSI). Une typographie européenne a aussi vu le jour sous le nom d'OCR-B par Adrian Frutiger [AdobeOriginals, nd]. Celle-ci plus naturelle est plus facile à lire pour l'homme.

3ème génération

En 1970, les obstacles de l'OCR sont les documents de mauvaises qualités, les imprimés larges et l'écriture manuscrite. C'est durant cette génération que l'Américain Ray Kurzwell crée son entreprise omni-font OCR, son logiciel de reconnaissance de police de caractère.

De nos jours

Dès les années 1990, les réseaux de neurones appliqués à la reconnaissance de caractères apparaissent avec les réseaux de neurones à perceptron multicouche [Sabourin and Mitiche, 1992] et les réseaux de neurones convolutifs [LeCun et al., 1995] permettent des avancées considérables. En 1998, la base de données MNIST [1] (*Modified National Institute of Standards and Technology database*) est créée. Elle permet d'analyser les performances de différents modèles car la base de données est assez grande pour représenter différents jeux de données réels. L'apprentissage automatique (*machine learning*) est rapidement apparu comme une solution efficace et c'est une des solutions les plus utilisées. A partir de la mise à disposition à la libre utilisation en 2005 de Tesseract [Smith, 2007], on peut dire que les systèmes d'OCR se sont démocratisés. Des tentatives d'adaptation à tous les types de formats ont explosé. Toutes les actions répétitives demandant l'oeil humain ont permis un éventail d'application de l'OCR. Nous avons des projets visant à la détection et la reconnaissance du texte dans des images ou des séquences vidéos [Wolf et al., 2001]. On fait des recherches sur de nombreuses langues comme Google

1. http://yann.lecun.com/exdb/mnist/

Drive qui permet l'océrisation de plus de 200 langues dans 25 systèmes d'écriture [Dmitriy Genzel, 2015]. Les entreprises les utilisent pour gagner en temps et en main d'oeuvre comme la reconnaissance des plaques d'immatriculation au Quatar [Farhat et al., 2016] mais aussi en France.

Date	Evènement
1870	Les premières tentatives
1940	La version moderne de l'OCR
1959	Les premières machines à OCR
1960-1965	1ère génération
1965-1975	2ème génération
1975-1985	3ème génération
1986 -	Démocratisation

TABLE 2.1 – Récapitulatif de la chronologie du développement de l'OCR

La technologie durant notre époque est mature. La précision des systèmes d'OCR pour les langues occidentales utilisant l'alphabet latin sur les textes imprimés atteint les 99%. La recherche se concentrent maintenant sur l'écriture manuscrite ainsi que les textes imprimés dans les langues orientales asiatiques et la préservation des ouvrages historiques qui reprennent toutes les difficultés de l'OCR à ses débuts mais avec des méthodes déjà éprouvées. Les recherches continuent donc dans ce domaine pour toujours apporter des améliorations. Ainsi, la segmentation qui a toujours été une étape importante d'un système d'OCR n'est pas nécessaire avec un système proposé par [Hasnat et al., 2007] utilisant le modèle caché de Markov pour le Bangla. Avec le même modèle caché de Markov, [Ait-Mohand et al., 2010] adapte un système d'OCR pour reconnaître plusieurs polices. Pour notre cas, il y a peu d'exemple et de recherches sur l'océrisation du dictionnaire mais il en existe comme ce dictionnaire de prononciation du letton [Strankale and Paikens, 2020].

De nombreux outils d'OCR de qualité variables sont donc disponibles sur le marché mais peu d'entre eux sont gratuits et libres d'utilisation [Patel et al., 2012]. Quelques OCR toujours d'actualité dans tableau 2.2

OCR	Caractéristiques		
	Création	Open source	Entraînable
ABByy	1989		
OmniPage Pro	1970		
OCR4all	2019	x	x
Tesseract	1985	x	x
OCRopus/OCRopy	2007	x	x
Kraken	2015	x	x

TABLE 2.2 – Systèmes et/ou logiciel OCR

2.4 Les difficultées de l'OCR

[Hamad and Mehmet, 2016] enregistre 8 difficultées de l'océrisation :
— complexité des différents niveau de plan d'une image
— contraste et luminosité

— résolution
— inclinaison de l'image
— dégradation et flouttage
— typographie
— langue
— quantité et emplacement du texte dans une image

2.5 Les étapes d'océrisation

Généralement, la reconnaissance d'un document s'effectue en quatre étapes hormis l'acquisition de l'image. La précision d'un système d'OCR repose principalement sur le pré-traitement effectué sur l'image ainsi que la segmentation [Bieniecki et al., 2007]. Si ces deux étapes ne sont pas menées à bien alors l'étape de reconnaissance à peu de chance de bien réussir.

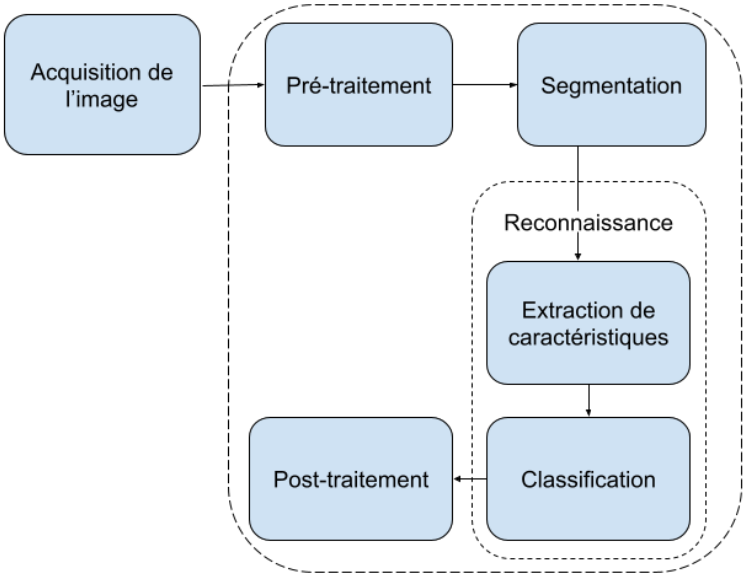


FIGURE 2.3 – Processus d'océrisation

2.5.1 Pré-traitement

La qualité de la numérisation joue un rôle important. On traite l'image afin de faciliter la reconnaissance. Elle peut prendre la forme d'une augmentation du contraste (binarisation pour une image en nuance de gris), ou d'un redressement du document (correction de l'orientation), ou d'un nettoyage de bruit. Un document numérisé peu à l'origine être de mauvaise qualité si celui-ci est un document historique. Dès lors, les pages peuvent ne pas être blanches, avoir des froissures.

2.5.2 Segmentation

La segmentation se fait à plusieurs niveau. La segmentation de la page et la segmentation du texte. Trouver les zones de texte, puis les lignes de textes, de mots et en dernier de caractères pour les isoler et ne pas être influencé par d'autres contenus en arrière plan autre que des caractères. Il y a trois catégories d'algorithmes pour la segmentation [Hamad and Mehmet, 2016].

— méthode descendante (*top-down methods*)
— méthode ascendante (*bottom-up methods*)
— méthode hybride (*hybrids methods*)

La technique de la projection d'histogramme horizontal et vertical [Shinde and Chougule, 2012] semble obtenir plus facilement la segmentation en lignes, mots et caractères.

2.5.3 La reconnaissance de caractères

C'est l'étape de l'extraction du texte à proprement parler. C'est durant cette étape qu'on va utiliser deux principaux types d'algorithmes pour la reconnaissance :

— la correspondance de motifs
— l'extraction de caractéristiques

La correspondance de motifs fonctionne en isolant une image de caractère qu'on appelle glyphe qu'on compare à un glyphe stocké de manière similaire. Elle ne fonctionne bien qu'avec les images numérisées de documents qui ont été tapés dans une police connue.

L'extraction de caractéristiques décompose les glyphes en caractéristiques telles que les lignes, les boucles fermés, la direction des lignes et les intersections de lignes. Il utilise ensuite les caractéristiques pour trouver la meilleur correspondance ou le plus proche voisin parmi ses différents glyphes stockés.

Il s'agit de reconnaître chaque caractères grâce à ses caractéristiques typographiques, ou en le comparant à une base.

2.5.4 Post-traitement

C'est la dernières étapes où nous faisons une correction des erreurs éventuelles sur les données textuelles qui ont été extraites en un fichier informatisé. Cette correction peut se faire manuellement à l'aide du *crowdsourcing*[Jovian and Amprimo, 2011].

2.6 Les techniques de ROC

[Rao et al., 2016] présente les techniques qui sont souvent utilisées dans l'OCR comme étant les suivantes :

Template matching

Template matching compare chaque caractère à une bibliothèque de matrices de caractère. Lorsqu'une image correspond à une matrice de pixels, il considère celui-ci comme l'image du caractère correspondant. La reconnaissance est la meilleur sur les pages ne possédant pas une structure multi-colonnes et est la plus sensible aux défauts

Techniques probabilistes

Les principales techniques probabilistes utilisés dans la reconnaissance par [Dongre and Mankar, 2011] sont le plus proche voisin, la classification bayésienne, le modèle de Markov caché, le raisonnement approximatif [Gowan, 1995], le classificateur quadratique

Extraction de caractéristiques

L'extraction de caractéristiques consiste à transformer arbitrairement des données comme des textes ou des images en des données numériques par la vectorisation. Pour l'extraction des caractéristiques d'un texte, on appelle ce processus la représentation en sac de mots. Mais en OCR, on va rechercher les caractéristiques par rapport aux pixels. Par exemple, il y a la technique de la distance profile (*distance profile*[Verma and Ali, 2012], la projection d'histogramme [Sharma et al., 2013]

Réseaux de neurones

Un réseau de neurones est une architecture d'un multitude de connexion parallèle de noeuds qui selon le renforcement de certaines connexions vont donner des réponses différentes. Dans le cas de la reconnaissance optique de caractères. nous avons un réseau de neurones à apprentissage supervisés car on présente au réseau des entrées c'est-à-dire l'image en même temps que les sorties que l'on souhaite, la transcription.

2.7 ROC, les avancements sur les sinogrammes

La reconnaissance des caractères chinois est différente comparé aux différentes langues occidentales. Les langues occidentales comme l'anglais, le français, l'allemand et bien d'autres ont des variations de leur système d'écriture mais leur écriture dérive du latin et possède donc un alphabet avec une quantité définie de caractères. Ce qui n'est pas le cas du chinois. La quantité de caractères existant n'est pas le seul problème, la complexité d'un caractère n'est aussi pas la même. On peut mesurer la complexité d'un caractère par le nombre de traits qu'il contient. En latin, un caractère s'écrit en 1 à 4 traits alors que les caractères chinois vont utiliser jusqu'à 36 traits. Les travaux sur les caractères du mandarin ont été entrepris en 1960 par R. Casey et G. Nagy dans le but d'obtenir un système qui puisse reconnaître plus de 1000 symboles dans une seule typographie et qui puisse être implanté à l'époque[Casey and Nagy, 1966], utiliser le radical et former des sous groupes étaient déjà une des méthodes pour la reconnaissance du mandarin. La reconnaissance se poursuit dans cette direction avec la reconnaissance des formes basé sur la description et l'analyse de la structure des caractères du mandarin dans un cadre pour savoir où sont placés les traits[Stallings, 1972]. Cette méthode reste d'actualité, [Zhang et al., 2018] choisit de faire une analyse des caractères du mandarin par décomposition. La typographie est aussi un des problèmes des OCR. Il existe énormément de polices pour le mandarin que ce soit pour le système d'écriture simplifié ou traditionnel. Des travaux comme celui de [Zhong et al., 2015] essaie de palier à ce problème en présentant un OCR pour de multiples typographies en mettant en place un réseau de neurones convolutif avec une couche de *pooling* avant la dernière couche de convolution et en mettant un peu de bruit dans une police. Cette déformation permettant de meilleurs résultats. Un autre point important est que la quantité de caractères existant n'étant pas des moindre et les caractéristiques plus nombreuses obligent donc aussi un temps de calcul plus important. [Zhang et al., 2013] propose d'améliorer ce temps de calcul par une classification SVM et de la technique du plus proche voisin durant l'étape de reconnaissance après l'extraction de caractéristiques. Dans la reconnaissance optique des caractères asiatiques, un déséquilibre des jeux de données n'est pas rare. Des caractères

peuvent exister mais ne plus être souvent utilisés et/ou ne pas exister de base. Ces caractères qui n'existent pas de base ont peut être été créés en assemblant plusieurs sinogrammes ensembles pour des occasions, une utilisation spécifiques comme à Taiwan. Cependant, on peut reformer ces caractères car les glyphes qui constituent le caractères, eux existent. Une segmentation par projection d'histogramme vertical. Remédier au déséquilibre de jeux de données par la fonction objective CTC sur attention [Feng et al., 2019] pourrait aider. Le système OCR kraken que nous utilisons et que nous détaillerons dans le chapitre 3, section 3.3 contient aussi la fonction objective CTC dans son architecture.

2.8 Mesures d'évaluation

Pour [Karpinski and Belaid, 2016], l'évaluation peut ne se faire que sur les étapes de la segmentation et de la reconnaissance de caractères pour évaluer les performances de l'OCR. L'idéal étant un modèle qui puisse reconnaître les caractères quelque soit la typographie, la taille du caractère, l'état de l'image dans lequel apparaît le caractère. Mais un tel système n'existant pour l'instant pas, il reste nécessaire de faire une évaluation pour sélectionner le mieux adapter ou même optimiser celui qui obtient de meilleurs résultats. L'étape de pré-traitement n'étant peut-être pas connu et non obligatoire si les images de base sont de bonnes qualitées.

Déterminer le taux d'erreur entre le résultat de l'océrisation (le texte de sorti reconnu par l'OCR) et la vérité de terrain (*ground truth*, le texte de référence qui a été manuellement retranscrit) permet

La distance de leveinshtein est une métrique de distance qui mesure la différence entre deux séquences de caractères, c'est le nombre minimum de caractères requis pour changer un mot ou une séquence en une autre.

Dans la figure 2.4, I (Insertion) correspond à l'ajout d'un caractère, S (Substitution) à la confusion et D (Delition) à la confusion en plus de la suppression. Les lettres "c" et h" ont été confondues et remplacées par un caractère : le "G".

Château	Chanteau	Chat eau	Chapeau	Gâteau
	^	^	|	|v
Original	*Ajout*	*Ajout*	*Confusion*	*Confusion & Suppression*

FIGURE 2.4 – Différents cas d'erreur de reconnaissance possible sur le mot "château" ref, [Belaïd and Cecotti, 2006]

Character Error Rate

Le CER (formule 2.1) concerne de nombreuses langues y compris les langues asiatiques tel que le mandarin, le japonais, le coréen. C'est la mesure qu'on va privilégier. Il calcule le taux d'erreur par caractères.

$$\text{CER} = \frac{\text{I+S+D}}{\text{N}} \tag{2.1}$$

D'après le résultat du CER, on considère :

— CER 1-2 % : bon résultat
— CER 2-10 % : résultat moyen

— CER 10% : résultat médiocre

Un CER avec un résultat d'environ 20% est considéré comme satisfaisant pour un texte contenant des caractères rares.

Cette mesure faisant une évaluation au niveau des caractères contrairement au mot est la plus adaptée à notre cas. Dans de nombreuses langues sinitiques, le système d'écriture ne comporte pas d'espace entre les mots. Nous sommes dans cette situation. Les sinogrammes se suivent, nous ne pouvons nous reposer que sur la connaissance des sinogrammes et de leur sens pour délimiter les mots. Il est donc difficile de faire une comparaison au niveau des mots alors qu'il n'y a pas de caractéristiques à part la ponctuation.

Word Error Rate

Le WER (formule 2.2) Il calcule le taux d'erreur par mot. Dans notre cas d'une langue asiatique, ce n'est pas la mesure qu'on va privilégier. La formule est la même mais le calcule ne se fait pas au même niveau.

$$\text{WER} = \frac{\text{I+S+D}}{\text{N}} \tag{2.2}$$

Le Rappel

Le rappel (formule 2.3) mesure le nombre d'éléments correctement étiquetés par le système (ce sont les caractères corrects qui correspond au caractères du résultat - les erreurs) rapporté au nombre d'éléments étiquetés dans la référence (ce qui correspond aux caractères de la transcription)[Karpinski and Belaid, 2016].

$$\text{Rappel} = \frac{\text{caractères corrects}}{\text{caractères transcription}} \tag{2.3}$$

Précision

La précision (formule 2.4) mesure le nombre d'éléments correctement étiquetés par le système (caractères corrects) rapporté au nombre total d'éléments étiquetés par le système (caractères de la sortie).

$$\text{Précision} = \frac{\text{caractères corrects}}{\text{caractères de la sortie}} \tag{2.4}$$

F-mesure

La F-mesure (formule 2.5) est la moyenne harmonique pondérée du rappel et de la précision. La valeur accordée à β permet de pondérer le rappel ou la précision, ou d'équilibrer les deux mesures (avec $\beta = 1$).

$$\text{F-mesure} = \frac{(1+\beta^2) \times \text{ précision } \times \text{ rappel}}{\beta^2 \times \text{ précision } + \text{ rappel}} \tag{2.5}$$

Il est possible de faire ces calculs avec l'outil open source ocrevalUAtion [2] qui a été développé pour automatiser certain de ces calculs en fournissant la transcription et la sortie de l'OCR qu'on a choisi ou avec KaMI (Kraken Model Inspector) -tools projetc [3].

2. https://github.com/impactcentre/ocrevalUAtion
3. https://github.com/KaMI-tools-project

2.9 Conclusion

Au terme de ce chapitre, nous avons pu voir l'ensemble du processus de reconnaissance optique de caractères. Le processus générale ne change pas vraiment. Seul les données que nous apportons ont le plus gros impact ainsi que les techniques utilisées. Les systèmes de reconnaissance de caractères sont comme des puzzles de techniques dont on essaie de trouver le meilleur assemblage pour résoudre les problèmes qui surviennent selon les langues. Énormément de sites en ligne offre des systèmes d'OCR pour directement océriser des documents ou des possibilités d'intégration de leur système mais il nous est impossible d'avoir accès à leurs données d'entraînement ou à un modèle basique ré-entraînable. Les systèmes d'OCR créés sont spécifiques à une utilisation. La plupart des systèmes d'OCR qui sont libres d'utilisation ne comporte pas de modèle basique du chinois. Cependant les recherches sur la reconnaissance optique de sinogrammes continuent et ce surtout sur l'écriture manuscrite.

CHAPITRE 3

DESCRIPTIONS DES OUTILS RETENUS

Sommaire

3.1 Introduction

Dans ce chapitre, nous allons tout d'abord aborder l'outil qui a été utilisé (eScriptorium et kraken). Le choix de l'outil d'océrisation s'est fait en premier sur l'accessibilité de celui-ci, des recherches et expériences déjà obtenues puis (de sa compatibilité par rapport à notre projet). Nous avons donc porté notre choix sur cet outil car il est open source et libre d'utilisation. kraken possède un dépôt de modèles pré-entraînés mais celui-ci n'est pas très fourni surtout pour les langues asiatiques. Il n'y a pas de modèle de base à disposition pour que l'on puisse ré-entraîner et il n'est pas possible de faire l'apprentissage d'un modèle sans transcriptions. Une partie du processus s'est fait sur eScriptorium (le pré-traitement et la segmentation) et la partie restante s'est faite avec les commandes de kraken sur le terminal (l'entraînement et la validation des modèles).

3.2 eScriptorium

eSciptorium [1] fait parti d'un des domaines de recherche principal du projet scripta [2], celui de la section numérique dont le but est de développer des outils comme eScriptorium pour la reconnaissance optique d'écriture et l'analyse de documents tout en se focalisant sur les cultures non alphabétiques et pré-typographique et ce afin de pourvoir aux besoins de chercheurs dans le domaine des humanités avec un outil que

1. https://escriptorium.fr/
2. https://www.psl.eu/en/scripta

l'on pourrait considérer de couteau suisse car il intègre une multitude de fonctions pour transcrire, annoter, traduire et publier des documents historiques. L'interface est construite sur et autour de kraken pour son système de reconnaissance optique de caractères.

3.2.1 Installation

L'installation d'eScriptorium se fait avec [3] ou sans [4] Docker en local.

3.2.2 Interface

L'interface permet d'avoir facilement accès aux différentes fonctionnalitées que nous verrons au chapitre 4 permettant de travailler avec une grande quantité de documents.

A la création d'un projet sur eScriptorium, il est possible de choisir la langue et le système d'écriture principal. Ainsi que le sens de lecture. Dans la figure 3.1, nous avons précisé que le texte serait des caractères chinois avec le système *bopomofo* et un sens de lecture de gauche à droite la plupart du temps.

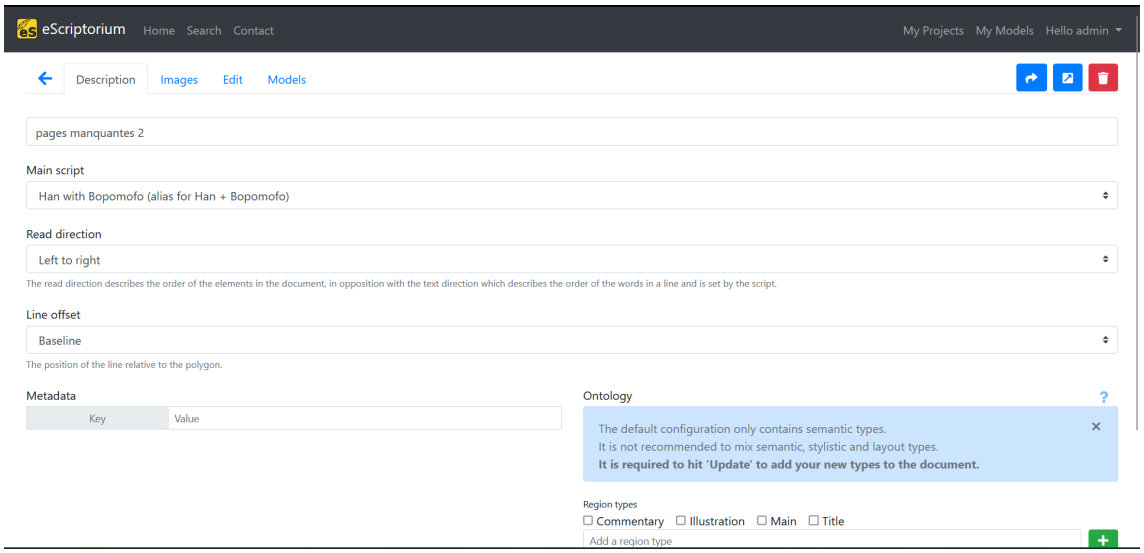


FIGURE 3.1 – Création d'un projet eScriptorium

Nous pouvons ensuite directement importer des fichiers pdf dont eScriptorium divisera toutes les pages contenu dans un fichier et les convertira en image au format png.

eScriptorium possède un modèle de base pour la segmentation qui donne un résultat correcte. Mais que nous avons quand même ré-entraîné. Nous avons travaillé avec l'interface et avec les commandes sur le terminal car un problème est survenu avec le fichier au format PAGEXML. Pour l'apprentissage d'un modèle, il nous est nécessaire de générer nous même les images ainsi que le fichier qui va de pair contenant les informations de localisation des zones de texte au format PAGEXML avec tout le contenu nécessaire. Or les fichiers PAGEXML que nous avons généré rencontraient un problème lorsque nous voulions entraîner un modèle de reconnaissance

3. https://gitlab.com/scripta/escriptorium/-/wikis/docker-install
4. https://gitlab.com/scripta/escriptorium/-/wikis/full-install

à partir de l'interface. L'entraînement commençait mais s'arrêtait brusquement. La commande d'entraînement directement lancée à partir du terminal ne rencontrait pas ce problème.

3.3 Kraken

Kraken [5], développé en Python et conçu pour être utiliser sur Linux est un fork retravaillé du projet OCRopus renommé OCRopy[Breuel, 2008]. Il reprend les principaux éléments et base d'OCRopus tout en apportant des modifications pour améliorer le résultat final. C'est un système optimisé pour les textes historiques et non latin car plus pensé pour les langues orientales. Ses principales caractéristiques permettent

— l'entraînement de la segmentation et de la reconnaissance de caractères
— la prise en compte de la direction du texte que ce soit de droite à gauche, de haut en bas ou qu'il y ait du texte dans des directions différentes
— la prise en charge des fichiers de formats ALTO/PageXML/abbyXML/hOCR
— un répertoire public des modèles disponibles
— des modèles dans un format léger
— une variété d'architecture de réseau pour la reconnaissance (VGSL)

La reconnaissance de caractères demande une multitude d'étapes pour son exécution complète. kraken n'y échappe pas. Il propose une analyse de la structure pour la segmentation de la page et extraire les lignes de texte d'une image, la reconnaissance qui va traiter les lignes de texte et les envoyer au classifieur et finalement la sortie des résultats dans un format approprié qui sont ALTO ou PAGEXML.

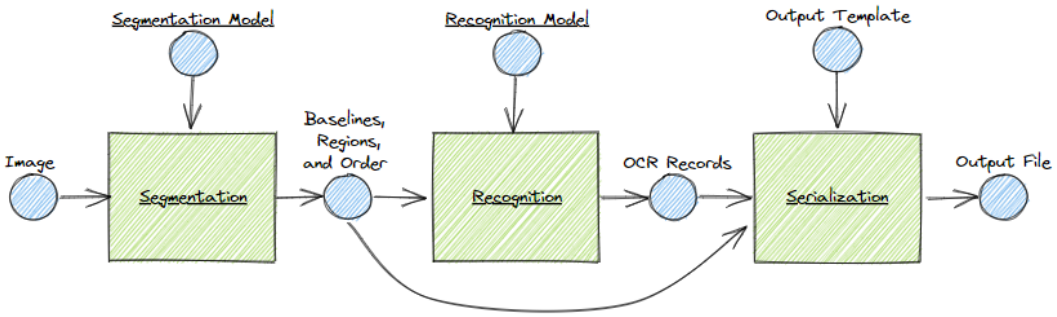


FIGURE 3.2 – Schéma montrant la modularité des commandes
ref, https://kraken.re/master/index.html

https://kraken.re/master/api.html nous détaille les étapes de son algorithme.

Segmentation

La méthode de segmentation de la *baseline* est basée sur un réseau de neurones qui classifie les pixels de l'image en *baseline* qui correspond à la ligne où il devrait y avoir du texte et en zone de texte. Le modèle de segmentation va contenir les informations définissant les types de lignes et de régions, les régions et une autre *baseline* pour le masque.

5. https://kraken.re/master/index.html

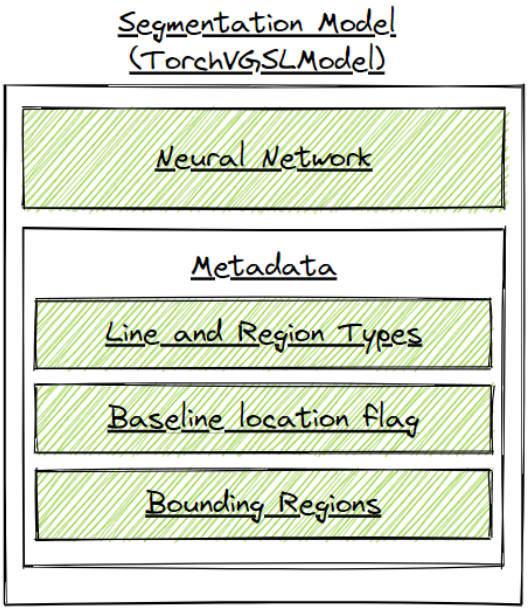


FIGURE 3.3 – Modèle de segmentation
ref, https://kraken.re/master/api.html

Reconnaissance

La reconnaissance est un processus en plusieurs étapes qui va produire une matrice avec une valeur de confiance pour chaque sortie à chaque étape. Cette matrice est décodé en une séquence de labels entiers qui sont mappés en un code unicode à l'aide d'un codec. Les labels et les points codes se correspondent, c'est-à-dire un label pour un point code Unicode. Mais il est possible si on le veut avec un décodeur plus complexe de mapper un label à une multitude de point code, l'inverse est aussi possible de même que le mappage de multiple codes à de multiples points. Le réseau de neurones va mapper l'image d'une ligne (la séquence d'entrée) en une séquence de caractères (la séquence de sortie) comme le montre la figure 3.5.

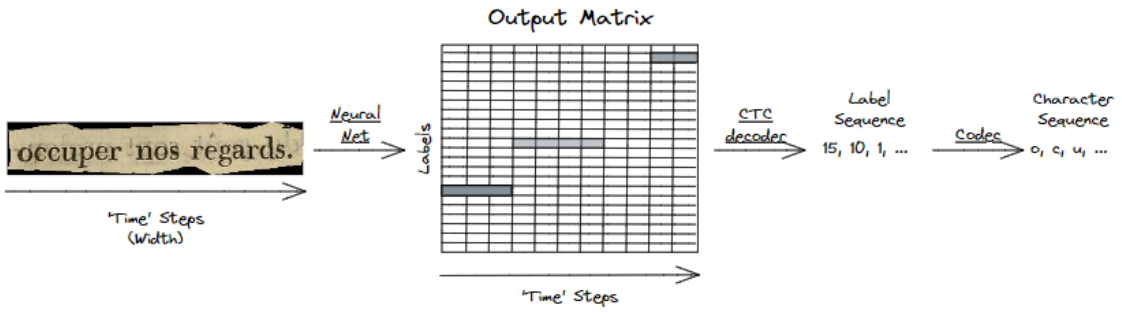


FIGURE 3.4 – Processus de reconnaissance d'une ligne
ref, https://kraken.re/master/api.html

Les codecs font partis du modèle et n'ont pas besoin d'être manuellement intégrés. Par contre, il est possible de choisir le décodeur CTC. Comme montré dans la fi-

gure 3.4, le modèle de reconnaissance combine le réseau de neurones, le codec, les informations de l'entrée pour savoir si c'est une image en nuance de gris ou binaire et le décodeur CTC pour la conversion en séquence de labels.

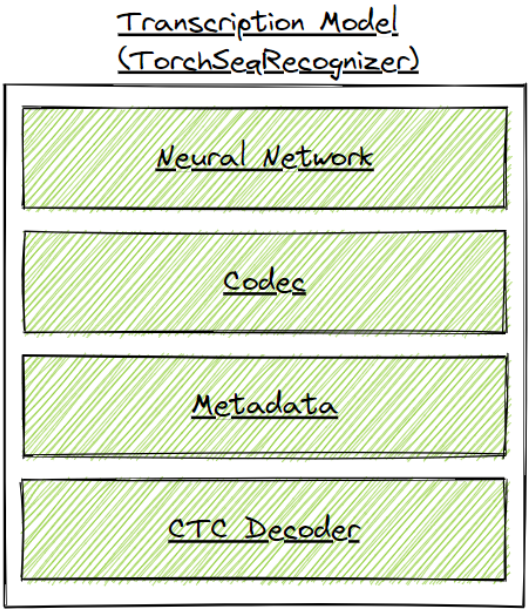


FIGURE 3.5 – Modèle de reconnaissance
ref, https://kraken.re/master/api.html

3.3.1 Installation

L'installation n'est pas compliquée. Nous pouvons facilement l'installer en tapant la commande d'installation de base dans le terminal.

```
pip install kraken
```

Cependant, il faut bien faire attention aux paramètres et à la version des différents modules uilisés pour le bon fonctionnement de kraken. Les modules comme torchvision qui servent à analyser les images doivent être dans la version appropriée par rapport à la carte graphique. Il est préférable de vérifier les paramètres de la carte graphique car l'installation des modules de kraken peuvent se passer sans encombre mais les commandes d'entraînements rencontreront une erreur après leur lancement.

```
Cuda error: no kernel image is available for execution
on the device
```

Nous avons réglé cette erreur en installant CUDA 11, la version installée était CUDA 10. De plus, l'installation ne peut actuellement pas se faire avec la version la plus récente de python, c'est à dire python 3.10 lors de l'écriture de ce mémoire.

3.4 Deux formats d'entraînement

Le réseau de neurones est un réseau hybride convolutionnel et un réseau de neurones récurrent entrainé avec la fonction objective CTC qui va réduire les données d'entraînement au niveau d'une ligne de transcription. L'architecture actuel permet

une intervention moindre de l'utilisateur[Kiessling, 2019]. Pour l'entraînement des modèles, nous avons gardé l'architecture de base de la figure 3.6 et n'avons modifié ni les paramètres ni les hyperparamètres par défaut.

	Name	Type	Params
0	net	MultiParamSequential	4.4 M
1	net.C_0	ActConv2D	288
2	net.Gn_1	GroupNorm	64
3	net.C_2	ActConv2D	16.4 K
4	net.Gn_3	GroupNorm	128
5	net.Mp_4	MaxPool	0
6	net.C_5	ActConv2D	73.9 K
7	net.Gn_6	GroupNorm	256
8	net.Mp_7	MaxPool	0
9	net.S_8	Reshape	0
10	net.L_9	TransposedSummarizingRNN	1.1 M
11	net.Do_10	Dropout	0
12	net.L_11	TransposedSummarizingRNN	1.6 M
13	net.Do_12	Dropout	0
14	net.L_13	TransposedSummarizingRNN	1.6 M
15	net.Do_14	Dropout	0
16	net.O_15	LinSoftmax	92.3 K

FIGURE 3.6 – Architecture du réseau

Le modèle de segmentation cherche les zones de texte et les *baselines* dans une image de page. Le modèle de reconnaissance convertit les images des lignes de texte repérées par le segmenteur en texte numérique. kraken accepte plusieurs formats de données d'entraînement :

— ALTO
— PAGEXML

3.4.1 ALTO

Le fichier au format ALTO[6] que kraken produit suit les recommandations de ALTO 4.2.

3.4.2 PageXML

Le fichier au format PAGEXML que kraken produit suit les recommendations de la version 2019-07-15 mais l'analyseur de structure est plus souple et peut laisser passer des structures moins conformes. La figure 3.7 représente le schéma de la structure PAGEXML, celle-ci avec juste les informations nécessaire à l'entraînement de kraken n'est pas simple à trouver.

Nous avons choisi de reproduire les fichiers de la structure PAGEXML au lieu d'ALTO car la structure PAGEXML nous semble plus simple à reproduire. Les informations qu'elle contient étant aussi moins détaillé mais bien suffisant pour l'entraînement. kraken conseille aussi d'entraîner le modèle de reconnaissance avec un format de fichier binaire pour de larges quantités de données. Cependant, nous somme resté sur la paire de fichiers image/PAGEXML car la commande train de kraken possède une option -t qui permet de travailler avec un fichier texte contenant la

6. https://www.loc.gov/standards/alto/techcenter/elementSet/index.html

```
<PcGts
xmlns="http://schema.primaresearch.org/PAGE/gts/pagecontent/2019-07-15"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://schema.primaresearch.org/PAGE/gts/pagecontent/2019-
07-15 http://schema.primaresearch.org/PAGE/gts/pagecontent/2019-07-
15/pagecontent.xsd">
        <Metadata>...</Metadata>
        <Page imageFilename="filename.jpg"...><!-- relative path to an
image file from the location of the XML document -->
                <TextRegion id="block_N"
                            custom="structure {type:region_type;}"><!--
region type is a free text field-->
                        <Coords points="10,20 500,20 400,200, 500,300,
10,300 5,80"/><!-- polygon for region boundary -->
                        <TextLine id="line_K">
                                <Baseline points="80,200 100,210,
400,198"/><!-- required for baseline segmentation training -->
                                <TextEquiv><Unicode>text text
text</Unicode></TextEquiv><!-- only TextEquiv tags immediately below
the TextLine tag are parsed for recognition training -->
                                <Word>
                                ...
                        </TextLine>
                        ....
                </TextRegion>
                <TextRegion id="textblock_M"><!-- for lines not
contained in any region. TextRegions without a type are automatically
assigned the 'text' type which can be filtered out for training. -->
                        <Coords points="0,0 0,{{ page.size[1] }} {{
page.size[0] }},{{ page.size[1] }} {{ page.size[0] }},0"/>
                        <TextLine>...</TextLine><!-- same as above -->
                        ....
                </TextRegion>
        </Page>
</PcGts>
```

FIGURE 3.7 – Schéma de la structure PAGEXML
ref, https://kraken.re/master/ketos.html

liste des fichiers xml en une seule fois alors que la commande compile n'a pas cette option.

3.5 Conclusion

eScriptorium possède une interface qui nous permet de faire quasiment toutes les étapes de l'océrisation. Le tutoriel que eScriptorium met à disposition regroupe toutes les fonctionnalités qui m'ont été nécessaires tant qu'il n'y avait pas de problèmes pour les faire marcher.

Deuxième partie

Expérimentations

CHAPITRE 4

CORPUS ET MODÈLES

Sommaire

4.1 Introduction

Dans ce chapitre nous allons préciser plus en détails les données textuelles qui sont à notre disposition pour mener à bien nos expérimentations, les différentes manipulations et pré-traitements qui ont été nécessaires pour générer les corpus d'entraînement. En effet, rappelons que nous avons une grande partie des données déjà en TEI mais que certaines pages sont manquantes, nous utilisons donc les données disponibles pour créer artificiellement des données d'entraînement afin d'être capable de transcrire les pages pour lesquelles nous n'avons que l'image.

Nous allons tout d'abord décrire les données de base que nous possédions et la structure de ces données. Par la suite, nous nous intéresserons à l'extraction des informations qui nous sont nécessaires depuis le format TEI qui nous permettra la génération des données pour construire différents corpus contenant des ensembles d'image. Ces ensembles variables sont des hypothèses qui ont pour but d'améliorer les résultats de l'océrisation.

4.2 Données de base

Les données de base que nous possédons actuellement sont les pages numérisées du dictionnaires au format pdf. Le dictionnaire se compose de 2 volumes dont les définitions de caractères s'étalent sur 2341 pages.

4.2.1 Structure dictionnaire

Le dictionnaire est compilé en deux volumes. Il contient près de 13000 sinogrammes, environ 60 000 entrées pour en tout presque atteindre 5 millions de caractères. Il fournit des informations comme la prononciation traditionnelle, la syntaxe, la définition dans les deux langues et les relations sémantiques. Le dictionnaire est structuré d'une manière standard pour les langues sinitiques, dans l'ordre de la transcription du *zhuyin fuhao* et par syllabe c'est-à-dire que la première entrée commencera par le phonème *bo* (ㄅ).

Dans la figure 4.1, nous avons surligné en bleu la liste des sinogrammes et des syllabes qui apparaissent dans la page. Entouré en rouge, nous avons les entrées syllabaires

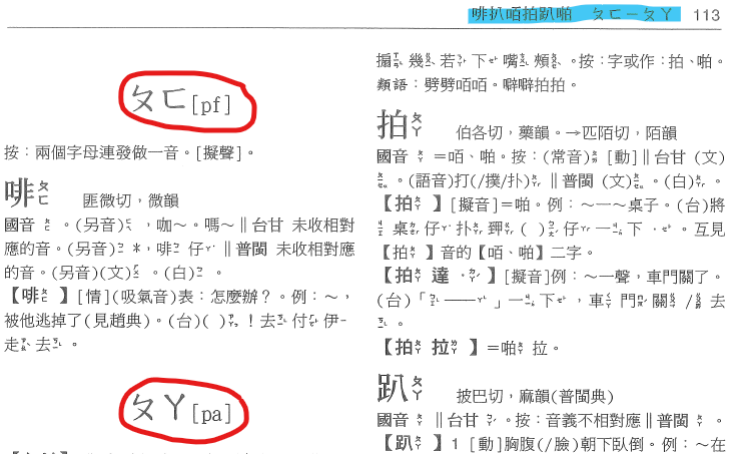


FIGURE 4.1 – Structure du dictionnaire

En dessous de chaque entrée syllabaire, nous avons la liste de sinogrammes correspondant à la syllabe ordonné selon le nombre de traits qui le compose. Si le nombre de trait est le même pour plusieurs sinogrammes alors c'est l'ordre du *zhuyin fuhao* qui prime. Dans la figure 4.2, nous avons entouré en gris une entrée sinogramme suivi en bleu de la syllabe en *zhuyin fuhao* et juste à côté, il y a la description phonologique. En jaune, nous avons les différentes prononciations. En rose, nous avons une numérotation car ce sont des homographes. Le caractère utilisé est le même mais pas le sens. En orange, nous avons la catégorie grammaticale qui est suivie de la définition en mandarin avec la traduction ou l'inverse. En vert, nous avons les synonymes et en violet, des caractères qui pourraient porter à confusion à l'écrit.

4.2.2 Structure TEI

La TEI (Text Encoding Initiative) est un projet universitaire pluridisciplinaire qui cherche à uniformiser autant que possible le codage de documents pour une meilleur exploitation en ligne ou hors-ligne [Ide and Véronis, 1996]. La structure que nous

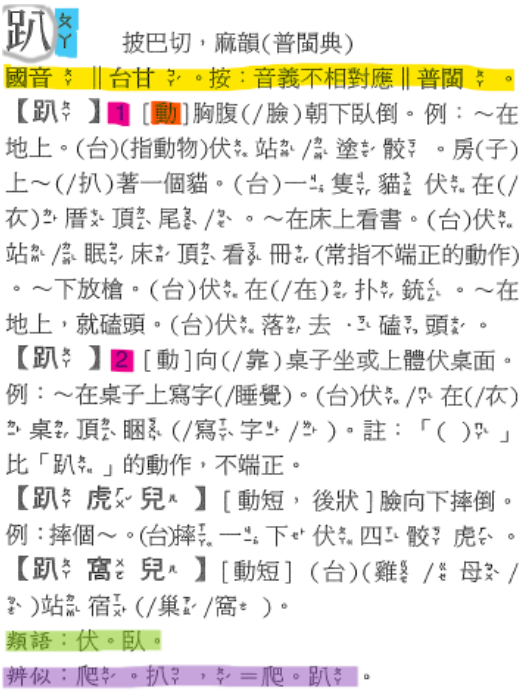


FIGURE 4.2 – Structure d'une entrée du dictionnaire

avions suivait les directives de la TEI pour les dictionnaires tout en adaptant à la structure de notre cas du dictionnaire. Le mapping de tous les caractères spéciaux ont été décrits dans la balise **teiHeader** dans des balises **charDecl** pour chaque caractère. En effet, l'auteur avait utilisé des milliers de glyphes spécifiques aux textes taiwanais et qui n'existaient pas dans l'encodage d'époque. Unicode n'existait alors pas. Il a été nécessaire de convertir à partir du Big5. La conversion s'est donc alors faite de l'encodage CP950 à de l'UTF-8 [1]. Ce mapping des caractères spéciaux s'est fait dans la zone à usage privé de Unicode et contient deux types de glyphes :

— des glyphes qui représentaient des transcription de syllabes en *zhuyin fuhao*, l'auteur avait toutes les variations
— des glyphes qui sont des sinogrammes spécifiques au taiwanais ou qui sont trop rares en mandarin si bien qu'ils n'ont pas été inclus dans le Big5 et ne sont parfois pas dans Unicode.

Dans la TEI, le *zhuyin fuhao* est décomposé, voir figure 4.4. Il faut aussi noter que certains caractères ne sont pas dans Unicode. Ainsi, ils apparaissent dans leur description IDS (*Ideographic Description Sequences*), figure 4.6.

FIGURE 4.3 – *Zhuyin fuhao* dans le dictionnaire

Pour le reste de la structure, des balises **entryFree** ont été utilisées pour les sinogrammes et des balises **entry** pour les mots.

1. https://github.com/g0v/koktai/blob/master/a-tsioh_sandbox/recode_utf8.pl

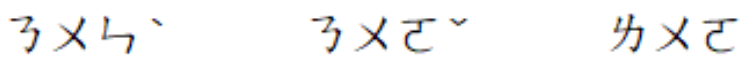
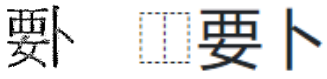


FIGURE 4.4 – *Zhuyin fuhao* décomposé

FIGURE 4.5 – Sinogramme du taiwanais et sinogramme rare



(a) original (b) décomposé (IDS)

FIGURE 4.6 – Caractère non inclus dans Unicode

Le fichier TEI est disponible sur Zenodo [2][吳守禮 , 2018]

4.3 Les pré-traitements

Les Pré-traitements ne concernent pas les images mais les données textuelles qui vont servir à la transcription et à la génération des corpus. C'est l'étape de la préparation des données. En effet, les pages scanner du dictionnaire étaient de bonne qualité. Nous n'avons pas fait face à toutes les problématiques que nous avons cités dans chapitre 2. Nous avons les données dans une structure TEI et il a fallu extraire seulement les données qui nous étaient nécessaires à la génération des images.

Dans la figure B.1, nous avons la structure complète de la TEI. Chaque colonne correspond à un niveau dans la TEI dans lequel on doit faire des extractions

	div type="sinograms"	**superEntry**	**div type="words"**
head **p**	**form** **orth** **g** contient le mapping **note[type="comment"]** **note[type="** 國音 **"]** **note[type="** 台甘 **"]** **note[type="** 普閩 **"]**	**orth [type="full"]** **def**	**orth[type="full"]** **def**

TABLE 4.1 – Équivalences de leurs dénominations dans les deux systèmes

Pour pouvoir retrouver les chemins vers les images des *zhuyin fuhao* et des sinogrammes dans les répertoires dans lesquels ils se trouvaient, nous avons nettoyé et balisé à l'aide d'expressions régulières. La référence des images étaient un peu différente et il y avait des informations superflues qui ne nous intéressaient pas. A l'origine, la référence était comme ça **ruby-FM3-ffd7d** et est devenu comme ça |ㄆㄧ **-m3/fd7d.png**|. Nous avons aussi rencontré des balises bizarre tel que pour **g**

2. https://zenodo.org/record/1308746#.Y2bksnbMK3A

ref="ruby –0" car il y avait des erreur dans l'impression du dictionnaire ou autre. Le *zhuyin fuhao* n'était pas présent à côté du caractère dont il devait définir la prononciation. A ce moment là on sautait ces références. Le texte que nous obtenons après l'extraction figure B.2.

Tout ce processus de pré-traitement et d'extraction s'est fait avec un script python que nous avons écrit ExtractAnnot.py, voir figure A.1.

4.4 L'entraînement de Modèle

4.4.1 Modèle de segmentation

Pour la segmentation des pages du dictionnaire, nous avons ré-entraîné le modèle de base que fournit eScriptorium. La segmentation que donne le modèle de base est correcte figure B.3. La raison de ce résultat doit être du à l'objectif de eScriptorium de travailler sur des anciens manuscrits dont la structure est parfois proche de celui de notre dictionnaire. Nous pouvons voir en vert la délimitation des textes. Dans cet exemple, les lignes vertes suivent bien le contour du texte mais les deux colonnes ne sont pas séparées. Cela impacte la longueur figure B.4 et l'ordre des lignes figure B.5. Les lignes numérotées 63, 69 et 70 s'étendent sur toute la largeur de la page et considèrent les textes dans les deux colonnes comme une seule ligne. De plus, nous avons un mélange de l'ordre des lignes suite à la direction de lecture que nous avons choisi à la création du projet sur eScriptorium. En faisant reconnaître la structure multi-colonnes, l'ordre des lignes va en priorité s'aligner aux lignes qui sont dans une même boîte.

Pour l'entraînement de la segmentation, nous avons travailler dans l'interface. Elle se présente sur une page et à chaque option que nous voulons, la page se divise verticalement pour laisser la place à une autre fenêtres figure B.6. A partir de cette fenêtre, nous avons manuellement refait la segmentation sur environ 200 pages. Nous avons délimiter par des boîtes les textes continus d'un sinogramme. Lorsqu'une entrée se terminait et laissait un peu d'espace blanc pour borner la fin d'un article et annoncer le début d'un nouveau sinogramme ou d'une nouvelle entrée syllabaire, nous avons fait de même. Nous obtenons quelque chose comme ça figure 4.7

La segmentation finale n'obtient un score que de 68,3 %. Il reste des erreurs d'ordre de lignes et parfois de *baseline* mais nous avons une amélioration de l'ordre dans les zones de texte. Une correction manuelle peu se faire en déplaçant des blocs de lignes en une seule action. Le résultat convient parfaitement. voir figure B.7, figure B.8, figure B.9.

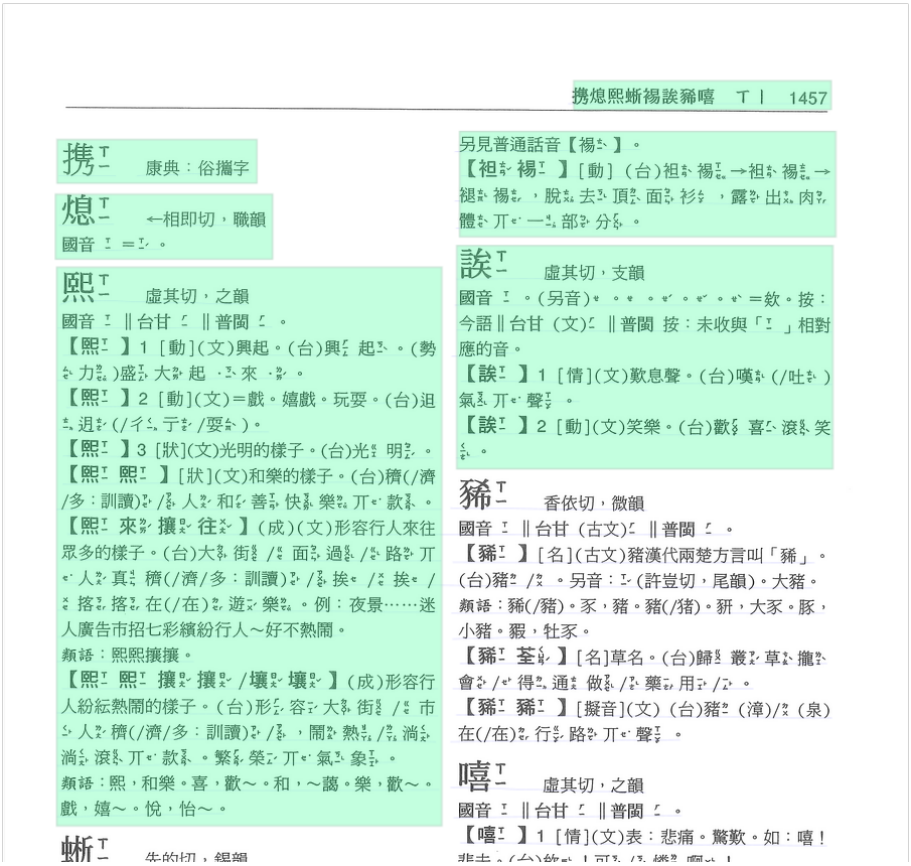


FIGURE 4.7 – Segmentation manuelle

4.4.2 Modèle de reconnaissance

Pour lancer l'apprentissage des modèles, nous avons la commande suivante :

Il ne faut pas oublier de spécifier la structure du format XML utilisé avec l' option -f.

```
ketos train -d cuda:0 -f page
```

Si on ne spécifie pas la structure, alors on rencontre l'erreur suivante :

```
RuntimeError : DataLoader worker is killed by signal: Killed
```

Pour entraîner un modèle, kraken nous laisse deux méthodes au choix :

— entraîner en utilisant une paire de fichiers image et texte

— entraîner en utilisant une paire de fichiers image et ALTO/PAGEXML

Nous sommes parti sur une paire de fichiers image et un fichier de structure PAGEXML. Dans la partie précédente, nous avons vu la structure d'un fichier PAGEXML. Dans l'exemple de structure, les coordonnées du masques ne sont obligatoires. Mais ce n'est pas le cas. Sans ces données, nous avons une erreur de structure du fichier PAGEXML. Le bon lancement pour l'entraînement d'un modèle suppose qu'on a au moins ces informations :

— le nom du fichier et la taille de l'image dans Page

— les coordonnées de la zoneT dans TextRegion

— les coordonnées du mask dans TextLine

— la baseline dans Baseline
— la transcription dans Unicode

Il est possible d'utiliser le module ocrd pour vérifier la structure de la page et les metadata. Mais on ne peut pas vérifier si les données qu'on a apporté pour la localisation des textes sont correctes. Pour le masque, kraken met à disposition un script **repolygonize.py**[3] pour le calculer. Mais on s'est rendu compte que le script n'arrivait pas à bien calculer le masque pour les images que nous avons généré et qu'il nécessitait trop de temps pour toutes nos images. Pour re-calculer le masque de toutes nos images, il fallait plusieurs jours. La plus grosse difficulté dans les données à donc été de fournir les coordonnées de la zone de texte et les coordonnées du masque qui sont de forme rectangulaire. Les cordonnées de ces éléments ne se listaient pas dans le même sens. Pour les coordonnées de la zone de texte, nous avons quatre coordonnées x,y où x et y sont des entiers. Ces coordonnées doivent se suivre en partant de la pointe haute à gauche dans le sens inverse des aiguilles alors que pour le masque, c'est le contraire. Nous allons dans le sens des aiguilles d'une montre. Nous pouvons ajouter à cela qu'étant donné que nous faisions des variations de taille de police, les coordonnées devaient donc s'adapter. Pour la *baseline*, nous avons fait la somme de la largeur des caractères et des images pour obtenir une *baseline* qui soit un peu plus longue que la longueur de la séquence de caractères. Nous avons essayé de faire varié la police à chaque génération d'une nouvelle image en faisant un choix aléatoire entre trois polices.

— uming.ttc
— ukai.ttc
— NotoSerifCJK-Regular.ttc

Nous avons généré tout cela avec le script GenerImg.py, voir figure A.2.

Pour rendre le système plus robuste, nous avons apporté quelques modifications que nous pensons ont amélioré les résultats :

— variation de la taille
— variation de la police
— variation de la *baseline*

4.5 Création des corpus

Pour créer les données qui vont servir à l'entraînement des modèles. Nous avons généré nous même les images et les fichiers PAGEXML correspondant. Nous avons essayé de rester le plus proche possible de la structure d'une page de dictionnaire au niveau de la ligne. Nous n'avons pas reproduit toute la structure visuelle d'une page de dictionnaire décrite un peu plus tôt, c'est-à-dire

— la pagination en haut à droite ou à gauche avec le *zhuyin fuhao* (注音符號) et les sinogrammes présents sur la même page, nous n'avons de plus pas ces informations
— deux colonnes de texte
— un titre centré au niveau d'une colonne

Les images que nous avons généré sont des images de tailles 1110x200, correspondant à peu près à la largeur d'une colonne du dictionnaire. Un résultat obtenu par la division de la taille des pages des fichiers pdf. Cette information concernant la taille

3. https://github.com/mittagessen/kraken/blob/master/kraken/contrib/repolygonize.py

est importante car la taille que nous percevons à l'oeil nu n'est pas la taille réelle des fichiers pdf. Chaque corpus est généré avec une variation du script **GenerImg.py**.

4.5.1 Corpus de base (modèle corpus_B)

Le corpus de base est la simple génération des toutes les lignes que nous pouvons. Le corpus contient 303 327 images.

4.5.2 Corpus de pages entière (modèle corpus_T)

À partir du fichier de transcription sans les chemins vers les images, nous avons ressorti un fichier texte ne contenant que les caractères. Ce fichier texte nous a permis de faire la transcription des pages originales dans l'interface d'eScriptorium. Mais ce travail de transcription reste chronophage. La durée de transcription est d'environ 25 minutes sans compter la réorganisation des lignes. Pour retranscrire une page entière, on devait faire un copier-coller à partir du fichier texte mais le fichier texte ne contient pas les séparations de page. On devait donc délimiter nous-même visuellement le contenu à copier. Puis il fallait re-découper le texte pour faire l'alignement à l'image. Cela engendrait une fatigue visuelle due à la taille et à la quantité de texte. Retranscrire toutes les pages originales n'était pas une option car le dictionnaire compte plus de 2000 pages. Mais il était nécessaire de transcrire des pages originales car le modèle entraîné sur les lignes ne permettait pas d'avoir un résultat concluant lorsqu'on testait et faisait une océrisation d'une page de dictionnaire. Nous avons retranscrit et intégré 5 pages.

4.5.3 Corpus moitié de données (modèle corpus50_T)

Pour ce corpus, les images qui ont été générées ont été réparties dans deux répertoires. Un répertoire contenant toutes les images paires et l'autre contenant toutes images impaires. Le modèle a été entraîné sur le répertoire paires avec des pages entières. Le corpus comptait 151 662 images.

4.5.4 Corpus des entrées de dictionnaire (corpus_TE)

Un autre corpus pour la différence de taille de police dans les entrées de dictionnaires. Nous pouvons voir que la taille de police pour le caractère d'entré est de taille différente comparée avec la taille de caractères qui suivent. Le corpus comptait 10 547 images. Nous avons donc re-généré des images pour ces lines spécifiquement. La différence de taille de la police

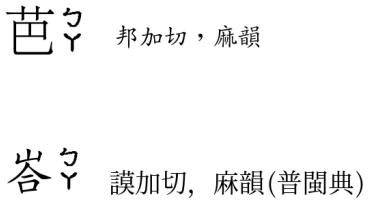


FIGURE 4.8 – Exemples des entrées

4.5.5 Corpus des caractères seuls (corpus_TS)

Nous avons généré des images de caractères seuls dans différentes polices pour que le modèle ait plusieurs occurrences des caractères. Surtout pour les caractères dont la fréquence était moindre dans le dictionnaire. Nous nous sommes aperçus que les polices ne contenaient pas tous les caractères bien qu'ils existent. Nous avons essayé de régler ce problème par une petite fonction qui devait vérifier si le glyphe existait dans le mapping de la police qui allait être utilisé. Si ce n'était pas le cas alors il le sautait. Cependant, parfois le code du caractère existait dans la police mais en réalité ce n'était pas le bon glyphe, voir figure 4.9. Le corpus comptait 10 247 images pour chaque police.

(a) référence (b) généré

FIGURE 4.9 – Caractère existant dans la police cjk portant le code U+22EF7
ref, https://graphemica.com/%F0%A2%BB%B7

4.5.6 Corpus de *bopomofo* (corpus_TZhuyin)

Dans ce corpus, nous avons voulu entraîner un modèle que pour reconnaître le *zhuyin fuhao*. Nous avons remplacé les caractères par des "." dans la transcription.

否則不成詞 ‖ 台甘(文)ㄨㄣˊ (語)ㄩˇ (漳)ㄓㄤ ˋ (廈、泉) ‖

```
<TextEquiv>
  <Unicode>..........ㄅㄚˊ...ㄅㄜˇ...ㄅㄨㄜˇ.....</Unicode>
</TextEquiv>
```

FIGURE 4.10 – Exemple d'une sortie pour le modèle corpus_TZhuyin

4.6 Conclusion

Dans ce chapitre nous avons pu détaillé le processus de génération des images d'apprentissage en présentant les données qui étaient à notre disposition, comment les préparer et les différents corpus qui vont permettre l'apprentissage des modèles.

CHAPITRE 5

RÉSULTATS

Sommaire

5.1 Introduction

Dans ce chapitre, nous allons discuter des résultats que nous avons obtenus à travers les différents corpus que nous avons généré pour répondre aux divers freins que nous avons observé. Nous avons toujours commencé par un corpus de base ne contenant que nos lignes générées auxquels nous y avons ajouté les autres corpus spécifiques. Il est nécessaire d'évaluer les résultats car l'*accuracy* supérieur à 90% qu'on obtient à la création d'un modèle n'est pas représentatif du résultat de l'océrisation d'une page du dictionnaire. Les résultats des évaluations ont été obtenu à l'aide du projet KaMI mentionné plus tôt dans le chapitre 2, à la fin de la section 2.8.

5.2 Résultats du corpus_B

Nous rappelons que le modèle de ce corpus ne s'est entraîné que sur les images des lignes que nous avons généré.

Le score CER est de 22,27%, nous sommes au-dessus du taux accepté pour considérer que le résultat soit satisfaisant pour un modèle faisant face à des caractères rares. Pour chaque page, le résultat est proche et peut dépasser le plafond des 20% que nous avons évoqué.

Nous avons un meilleur rappel par rapport à la précision même si la différence de 0,05 point n'est pas énorme. Cela signifie que l'OCR réussit à reconnaître la structure des textes mais n'arrive pas à associer les bons caractères.

Corpus évaluation	Caractères reconnus	I	D	S	CER(%)
Page 1	1092/1415	7	112	211	23.32
Page 2	1324/1759	5	141	294	25.01
Page 3	1657/2065	3	152	256	19.90
Page 4	1512/1994	3	192	290	24.32
Page 5	1648/2025	4	136	241	18.81
Total	7233/9250	22	733	1292	22.27

TABLE 5.1 – Résumé détaillé des résultats avec CER pour le corpus_B

Expérience	Rappel	Précision	F-mesure
Page 1	0.58	0.54	0.56
Page 2	0.54	0.50	0.52
Page 3	0.65	0.60	0.62
Page 4	0.57	0.51	0.54
Page 5	0.67	0.62	0.65
Moyenne	0.60	0.55	0.58

TABLE 5.2 – Précision, rappel et f-mesure pour le corpus_B

5.3 Résultats du corpus page originale

Corpus50_T

Nous rappelons que pour ce modèle, l'entraînement s'est fait sur la moitié des images générées. Ce sont les images du répertoire pair. Cela représente une ligne sur deux du dictionnaire. En plus, nous y avons ajouté cinq pages du dictionnaire.

Corpus évaluation	Caractères reconnus	I	D	S	CER(%)
Page 1	1185/1415	4	60	170	16.54
Page 2	1462/1759	6	75	222	17.22
Page 3	1775/2065	4	79	211	14.24
Page 4	1680/1994	9	92	222	16.20
Page 5	1752/2025	9	70	203	13.92
Total	7854/9250	32	376	1028	15.62

TABLE 5.3 – Résumé détaillé des résultats avec CER pour le corpus50_T

Le score du CER est assez bon dans l'ensemble. Nous obtenons une moyenne de 15.62 %. Le taux d'erreur le plus bas est celui de la page 5 avec 13.92 et le taux le plus haut pour ce modèle est de 17.22. Nous avons une différence de plus de 4 points. Cette différence se reflète aussi dans le nombre de caractères présents dans la page. A l'exception de la page 2 avec 1759 caractères, ce sont les pages qui contiennent le plus de caractères qui ont le plus faible taux d'erreur. Surtout pour les pages qui ont plus de 2000 caractères.

Il n'y a pas une grande différence entre le rappel, la précision et la f-mesure. Les scores sont corrects avec des résultats tournant autour de 0.70. La précision est un peu plus élevée que le rappel, cela signifie que le modèle n'a pas réussi à reconnaître tout le texte de l'image mais que les caractères reconnus sont en grande partie correct.

Expérience	Rappel	Précision	F-mesure
Page 1	0.67	0.70	0.66
Page 2	0.66	0.68	0.67
Page 3	0.71	0.74	0.72
Page 4	0.68	0.71	0.69
Page 5	0.72	0.75	0.73
Moyenne	0.69	0.72	0.69

TABLE 5.4 – Précision, rappel et f-mesure pour le corpus50_T

Corpus_T

Nous rappelons que le modèle s'est entraîné sur toutes les images de lignes générées en plus de cinq pages du dictionnaire.

Corpus évaluation	Caractères reconnus	I	D	S	CER(%)
Page 1	1168/1415	4	75	172	17.74
Page 2	1381/1759	12	86	292	22.17
Page 3	1733/2065	4	82	250	16.27
Page 4	1606/1994	5	120	268	19.71
Page 5	1727/2025	8	94	204	15.11
Total	7615/9250	33	457	1186	18.18

TABLE 5.5 – Résumé détaillé des résultats avec CER pour le corpus_T

Au niveau du CER, les meilleurs résultats sont pour les pages 3 et 5 avec 16.27% et 15.11% respectivement.

Expérience	Rappel	Précision	F-mesure
Page 1	0.68	0.65	0.66
Page 2	0.59	0.56	0.57
Page 3	0.70	0.68	0.69
Page 4	0.64	0.61	0.63
Page 5	0.73	0.70	0.72
Moyenne	0.67	0.64	0.65

TABLE 5.6 – Précision, rappel et f-mesure pour le corpus_T

Nous avons un rappel que nous pouvons considéré comme correct avec 0.67.

Nous pouvons voir à travers les résultats que le modèle de base avec les pages entières nous donnent de moins bons résultats. Le CER entre les deux modèles a une différence de plus de 2 points. Le modèle avec moins de données d'entraînement obtient une meilleur score avec 15.62% de taux d'erreur. Cette différence se voit aussi avec la f-mesure qui a subi une petite baisse.

5.4 Résultat du corpus50_TE

Nous rappelons que pour ce modèle, nous avons utilisé les images du répertoire pair en plus de cinq pages du dictionnaire, et nous y avons ajouté les lignes des entrées de sinogrammes.

Corpus évaluation	Caractères reconnus	I	D	S	CER(%)
Page 1	1158/1415	6	83	174	18.59
Page 2	1464/1759	3	97	198	16.94
Page 3	1770/2065	1	104	191	14.33
Page 4	1651/1994	5	133	210	17.45
Page 5	1755/2025	8	85	185	13.73
Total	7798/9250	23	502	958	14.21

TABLE 5.7 – Résumé détaillé des résultats avec CER pour le corpus50_TE

Le modèle obtient un très bon taux d'erreur de seulement 14,21%. Nous avons toujours une tendance où les pages contenant plus de caractères obtiennent un meilleur résultat.

Expérience	Rappel	Précision	F-mesure
Page 1	0.67	0.63	0.65
Page 2	0.70	0.66	0.68
Page 3	0.75	0.71	0.73
Page 4	0.70	0.65	0.67
Page 5	0.76	0.73	0.74
Moyenne	0.72	0.68	0.69

TABLE 5.8 – Précision, rappel et f-mesure pour le corpus50_TE

Nous avons des scores assez bon pour ce modèle. Un meilleur rappel par rapport à la précision.

5.5 Résultats du corpus50_TS

Pour ce modèle avec un seul caractère par image, nous avons voulu remédier au manque d'exemple pour les sinogrammes qui ont une fréquence d'apparition moindre par rapport à des caractères plus communs. Nous avons un modèle qui a pour base les corpus "pages entières", "moitié" et "caractère seul". Les polices que nous avons intégré sont :

— arphic-bkai00mp/bkai00mp.ttf
— arphic-bsmi00lp/bsmi00lp.ttf
— arphic-gbsn00lp/gbsn00lp.ttf
— arphic-gkai00mp/gkai00mp.ttf

Corpus évaluation	Caractères reconnus	I	D	S	CER(%)
Page 1	1151/1415	6	90	174	19.08
Page 2	1399/1759	7	105	255	20.86
Page 3	1731/2065	2	101	233	16.27
Page 4	1618/1994	8	124	252	19.26
Page 5	1714/2025	10	88	223	15.85
Total	7613/9250	33	508	1137	18.26

TABLE 5.9 – Résumé détaillé des résultats avec CER pour le corpus50_TS

Nous avons un rappel et une précision très proche pour ce modèle. il n'y a que 0.03 point de différence. Nous pensions que l'ajout d'images ne contenant qu'un seul

Expérience	Rappel	Précision	F-mesure
Page 1	0.66	0.62	0.64
Page 2	0.62	0.59	0.60
Page 3	0.71	0.67	0.69
Page 4	0.66	0.62	0.64
Page 5	0.71	0.69	0.70
Moyenne	0.67	0.64	0.66

TABLE 5.10 – Précision, rappel et f-mesure pour le corpus50_TS

caractère dans différentes polices aurait permis une meilleur performance mais ce n'est pas le cas. Nous n'avons pas présenté les résultats ici mais nous avons essayé d'entraîner un modèle avec encore plus de polices. Le modèle n'a pas du tout abouti car à la fin de l'entraînement, le modèle avait une reconnaissance négative avec un *accuracy* de -2,9% sur les données artificielles que nous généré.

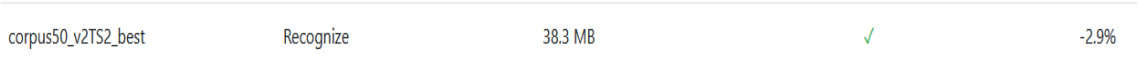


FIGURE 5.1 – Modèle corpus50_TS2

5.6 Résultats du corpus_TZ

Pour le corpus du *zhuyin fuhao*, il nous est impossible d'évaluer le modèle correctement. Visuellement, les résultats semblent plutôt corrects, voir figure 5.2. Etant donné que nous avons voulu créer un modèle qui ne puisse reconnaître que les *zhuyin fuhao* en remplacement la présence des autres caractères par des points, le *fine tuning* du modèle déjà possédé n'est pas possible. Les nouvelles caractéristiques d'un point représentant un caractère redéfini les règles de correspondances entre les caractères dans l'image et la transcription. Pour utiliser les résultats de ce modèle, il faudrait un autre modèle dans la logique inverse du modèle *zhuyin fuhao* pour fusionner les deux résultats. La fusion des résultats se feraient par une comparaison des deux sorties et un remplacement des points dans la sortie *zhuyin fuhao* par les caractères de la sortie de l'autre modèle.

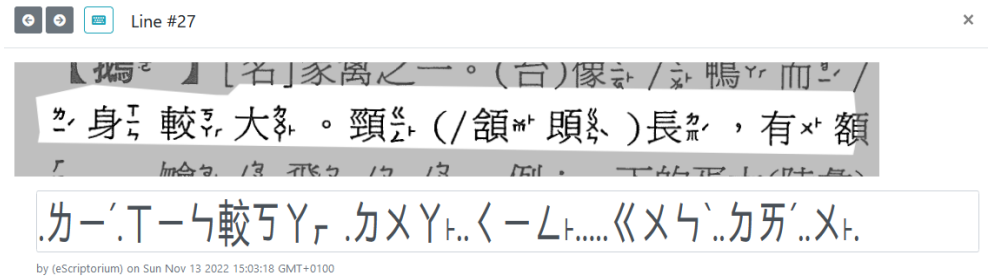


FIGURE 5.2 – Comparaison d'une ligne entre l'image et la sortie

Corpus évaluation	Caractères reconnus	I	D	S	CER(%)
Corpus_B	7233/9250	22	733	1292	22.27
Corpus50_T	7854/9250	32	376	1028	15.62
Corpus_T	7615/9250	33	457	1186	18.18
Corpus50_TE	7798/9250	23	502	958	14.21
Corpus50_TS	7613/9250	33	508	1137	18.26

TABLE 5.11 – Comparaison des résultats détaillés avec CER entre les modèles

Expérience	Rappel	Précision	F-mesure
Corpus_B	0.60	0.55	0.58
Corpus50_T	0.69	0.72	0.69
Corpus_T	0.67	0.64	0.65
Corpus50_TE	0.72	0.68	0.69
Corpus50_TS	0.67	0.64	0.66

TABLE 5.12 – Comparaison des moyennes des résultats de précision, rappel et f-mesure entre les modèles

5.7 Globalement

A travers les résultats obtenus des différents modèles entraîné, nous pouvons voir qu'il y a une tendance globale en rapport avec le nombre de caractères que contient une page d'évaluation. Pour la plupart des modèles, les pages ayant une quantité plus importante de caractères sont sur lesquels les scores sont les meilleurs. Ces pages contiennent tous plus de 2000 caractères avec la page 3 contenant 2065 et la page 5 contenant 2025. Une quantité plus importante de données ne permet pas forcément une amélioration significative si ce n'est une amélioration. Il n'y a pas de différence notable des les résultats du rappel, de la précision et de la f-mesure pour démarquer un modèle d'un autre. Cependant, les résultats du taux d'erreur par caractère montre une nette différence entre les modèles. C'est le modèle corpus50_TE qui sort du lot avec un taux d'erreur de 14,21%. Les erreurs les plus présentes sont la substitution et la delition de caractères. Les sinogrammes qui demande énormément de traits pour s'écrire sont évidemment les plus compliques à reconnaître. Normalement, les résultats devraient être un peu meilleur car comme nous l'avons évoqué dans la génération des images, certaines information et représentations qui sont dans le dictionnaire n'ont pas été ajouté aux données d'entraînement ou n'apparaissent pas assez souvent. Ces informations sont sans doute la cause de trois type d'erreur que nous retrouvons souvent dans les sorties :

— la numérotation des pages
— les caractères latin
— les entrées syllabaires
— les caractères décomposés

Nous avons avec la figure 5.3 ci-dessous une comparaison des résultats entre la transcription de la référence et la sortie de l'océrisation par l'outil KaMI. La colonne du milieu représente une fusion des deux textes dont les caractères sont dans différentes couleurs pour souligner quelles sont les différences. Le code couleur est le suivant :

— VERT : correct
— BLEU : insertion

— ROUGE : substitution/delition

Cette comparaison se fait sur les résultats du modèle corpus50_T. Nous avons un autre exemple avec des résultats médiocres.

REFERENCE	COMPARAISON	PREDICTION
哦噢噗哦噢喔噢哦阿 ㄛˊ-ㄜ 2313 抓了一大把出來。(台)哇ㄨㄚˊ啊·ㄚ!又ㄧㄨㄣ更(/復: 訓讀)ㄍㄜ/ㄍㄜㄣ 抄ㄙㄚ一ㄐㄧㄝ 大ㄉㄨㄚㄣ堆ㄉㄨㄧ出ㄘㄨㄝ 去·ㄎㄧㄤ。 【哦ㄛˊ喲·ㄧㄛ】[情]表:驚歎。例:~!這麼些。 (台)咻ㄍㄨㄚ!皆ㄐㄧㄚㄣ 恁ㄉㄧㄣ稽(/濟/多:訓讀)ㄗㄝˊ/ㄗㄨㄝㄣ。 噢ㄛˊ 反切見「噢ㄛ」項內 國音ㄛˊ。(另見)ㄛ、ㄛˇ、ㄏㄨㄛˋ。 【噢ㄛˊ】[情]表:半信半疑。例:~!他敢這樣兒(北方)。(台)哈ㄏㄚˊ啊ㄚㄣ ,伊ㄧ敢ㄍㄇˋ按ㄋ爾ㄋㄧ→ㄋㄝ? ! ㄛˇ [[o']] 噢ㄛˇ 反切見「噢ㄛ」 國音ㄛˇ。(另音)ㄏㄨㄛˋ。ㄛ、ㄛˊ、ㄛˇ// 台甘ㄏㄧㄝ; (罕用) ‖普閩ㄏㄧㄝ; ; 【噢ㄛˇ】[情]表:疑訝驚訝、疑惑。例:~,這道題怎麼還算不對?(台)咻ㄍㄨㄚˊ!此(/三此一)ㄐㄧㄤ 條ㄉㄧㄠˊ題 ㄉㄨㄝˊ/ㄉㄝˊ目ㄅㄛ; 按ㄋˋ怎ㄗㄨㄚˋ猶ㄧㄠˋ→ㄧㄠ算ㄙㄞㄟ膾ㄅㄝㄐㄅㄨㄝㄕ出ㄘㄨㄝ來 · ㄌㄞˊ。例:~!哪有這種事?(台)哈ㄏㄚㄣ 啊ㄚˋ!哪ㄋㄚˋ有 ㄨㄕ此(/三此一)ㄐㄧㄤ 號ㄏㄛㄕ載ㄉㄞㄕ(/戴ㄉㄞㄟ)事ㄐㄧㄟ(/事ㄉㄞㄕ情ㄐㄧㄟ)? ㄛˇ [[o']] 哦ㄛˋ 反切,見【哦ㄛˊ】項內。 國音ㄛˋ。(另音)ㄜˊ,吟~。ㄛˊ// 台甘(文)ㄍㄛˊ,ㄇㄛˊ ‖普閩ㄍㄛˊ。(俗音)ㄛˊ。 【哦ㄛˋ】[情]表:領會、醒悟。(廈)誠ㄛㄟ(普閩典)。例:~,我明白了。(台)喃ㄏㄛㄕ 喔·ㄛ,我ㄍㄨㄚˋ 知ㄗㄞ(/知ㄗㄞ影ㄧㄚˋ)喇(/啦)ㄌㄚㄟ。異文:噉ㄜ(陸美)。 噢ㄛˋ 反切見【噢ㄛ】音項內 國音ㄛˋ。(另見)ㄛ、ㄛˊ、ㄛˇ、ㄏㄨㄛˋ。 【噢ㄛˋ】[情]表:應答。例:~,是噢·ㄌㄡ!(陸美)。(台)哈ㄏㄚㄕ /ㄏㄚㄟ啊ㄚㄟ有ㄨㄕ影ㄧㄚˋ按ㄋㄟ爾·ㄋㄧ喃 ㄏㄛㄕ 。 類語:欸ㄟˋ。誤ㄝˋ。吼(/哼)ㄇ。呸(/嗯)ㄇ。 啊。 噯。□□□□,,,,。 ·ㄛ [[·o]] 喔 ·ㄛ 國音·ㄛ。(台甘)、(普閩)的反切——互見【	國哦膦噢膦噢酸哦膦噢釋喔膦噢照哦兒阿、三ㄛˊ一-ㄜ 2s3i1g3 如抓了一大把出來。(台)哇ㄨㄚㄚˊ啊·ㄚ!又ㄧㄨㄣ更(/復: 訓讀)ㄍㄜ/ㄍㄜㄣ 圾抄ㄙㄚ一ㄐㄧㄝ 大ㄉㄨㄚㄣ堆ㄙㄉㄨㄚㄧ出ㄘㄨㄝ 去·ㄎㄧㄤ。 【腰哦ㄛˊ腳喲。·ㄧㄡㄛ】[情]表:驚敵歎。 例:~[!這麼些。 (台)外咻ㄍㄨㄚ!皆ㄐㄧㄚㄣ 恁ㄉㄧㄣ稽(/濟/多:訓讀)ㄗㄝˊ/ㄗㄨㄝㄣ。 陽ㄅㄛˊㄛˇ 反切見「陽噢ㄛ」項內 國音ㄛˊ。(另見)ㄧㄛ~、ㄛˇ、ㄏㄨㄛˋ。 【膦噢ㄛˇ】[情]表:半信半疑。例:~!他前敢這壞樣 兒(北方)。(台)哈ㄏㄚㄚˊ啊ㄚㄣ ,伊ㄧ敢ㄍㄇˋ按,ㄋ爾ㄋㄧ→ㄋㄝ? 」! 字ㄛㄩˇ [[o']] 醫噢ㄛㄛ ˙ 反切見「噢喉ㄛㄛ」 國音ㄛˇ。(另音)ㄏㄨㄚㄛˋ。ㄛ、ㄛˊ、ㄛˇ// 台甘ㄏㄧㄝ; (罕用) ‖普閩ㄏㄧㄝ; ; 【陽噢ㄛˇ】[情]表:疑說訝驚語訝、疑惑。例:~,這 道題怎麼還算不對?(台)咬咻ㄍㄍㄨㄚˊ!此(/三此一)ㄐㄧㄤ 條ㄉㄧㄠˊ黑題 ㄉㄨㄝˊ/ㄉㄝˊ目ㄅㄛ; 按ㄋˋ怎ㄗㄨㄚˋ猶ㄧㄠˋ→ㄧㄠ算ㄙㄞㄟ膾ㄅㄝㄐㄅㄨㄝㄕ出ㄘㄨㄝ來。· ㄌㄞˊ。例:~!哪有這種事?(台)哈ㄏㄚㄚˊ啊ㄚˋ!哪ㄋㄚˋ有 ㄝㄨㄕ此(/三此一)ㄐㄧㄤ 號ㄏㄛㄕ載ㄉㄞㄕ(/戴ㄉㄞㄟ)事ㄐㄧㄟ(/事ㄉㄞㄕ情ㄐㄧㄟ)? ㄛ兵ㄇ`[[o']] 喉腰ㄛㄛ 反切,見【暗哦ㄛˊ】項內。 【國音ㄛˋ。(另音)ㄜˊ,吟~。ㄛˊ// 台甘(文)ㄍㄍㄜˊ,ㄇㄛˊ ‖普閩ㄍㄠㄛˊ。(俗音)ㄛㄛˊ。 【啞哦ㄛˋ】[情]表:領會、醒悟。(廈)說誠ㄝㄛㄟ(普閩 典)。例:~,我明白了。(台)喃ㄏㄛㄕ 喔·ㄧㄛ,我ㄍㄨㄚˋ 知ㄗㄞ(/知ㄗㄞ影ㄧㄚˋ)喇(/啦)ㄆㄌㄚㄟ。異文:噉ㄜ(陸美)。 噢ㄛˋ。 反切見【噢ㄛ】音項韻內 國音ㄛˋ。(另見)ㄛ、ㄛˇ:、ㄛˇ、ㄏㄨㄛˋ。 【陽噢ㄛˇ】[情]表:廟應答。例:;~,是噢嗌·ㄍㄌㄢㄡ!(陵陸美)。(台)哈ㄏㄚㄕ /ㄏㄚㄟ啊ㄇㄟㄌ)!有ㄨㄕ影ㄧㄚˋ按ㄋㄟ爾·ㄋㄧ頭喃 ㄏㄛㄕ 。	國膦膦酸膦釋膦照兒、三ㄛ一ㄛsig 如了一大把出來。(台)哇ㄨㄚˊ啊·!又ㄧㄨㄣ更(/復: 訓讀)ㄍㄜ/ㄍㄜㄣ 圾ㄙㄚ一ㄐㄧㄝ 大ㄉㄨㄚㄣ堆ㄙㄨㄚ出ㄘㄨㄝ 去·ㄎㄧㄤ。 【腰ㄛˊ腳。·ㄧㄡ】[情]表:驚敵。例:~[這麼些 (台)外ㄍㄨㄚ!皆ㄐㄧㄚㄣ 恁ㄉㄧㄣ稽(/濟/多:訓讀)ㄗㄝˊ/ㄗㄨㄝㄣ。 陽ㄅㄛˊ 反切見「陽ㄛ」項內 國音ㄛˊ。(另見)ㄧ~ㄛˇ、。 【膦ㄛˇ】[情]表:半信半疑。例:~!他前這壞 兒(北方)。(台)哈ㄏㄚㄚˊ啊ㄚㄣ ,伊ㄧ敢ㄍㄇˋ按,爾ㄋㄧ→ㄋㄝ? 」 字ㄩ [o']] 醫ㄛ 。反切見「喉ㄛ 國音ㄛˇ。(另音)ㄏㄨㄚˋ。ㄛ、ㄛˊ、ㄛˇ// 台甘ㄏㄧㄝ; (罕用) ‖普閩ㄏㄧㄝ。 【陽ㄛˇ】[情]表:疑說驚語、疑惑。例:~,這道題怎麼還算不對?(台)咬ㄍㄨㄚˋ!此(/此一)ㄐㄧㄤ 條ㄉㄧㄠˊ黑 ㄉㄨㄝˊ/ㄉㄝˊ目ㄅㄛ; 按ㄋˋ怎ㄗㄨㄚˋ猶ㄧㄠˋ→ㄧㄠ算ㄙㄞㄟ膾ㄅㄝㄐㄅㄨㄝㄕ出ㄘㄨㄝ。 來。 ㄌㄞˊ。例:~!哪有這種事?(台)哈ㄏㄚㄚˊ啊ㄚˋ!哪ㄋㄚˋ有 ㄝㄕ此(/三此一)ㄐㄧㄤ 號ㄏㄛㄕ載ㄉㄞㄕ(/戴ㄉㄞㄟ)事ㄐㄧㄟ(/事ㄉㄞㄕ情ㄐㄧㄟ)? 兵ㄖˋ[']] 腰ㄛ 反切,見【暗ㄛˊ】項內。 【國音ㄛˋ。(另音)ㄜˊ,吟~。ㄛˊ// 台甘(文)ㄍㄛˊ,ㄇㄛˊ ‖普閩ㄍㄠㄛˊ。(俗音)ㄛˊ。 【啞ㄛˋ】[情]表:領會、醒悟。(廈)說ㄝㄛㄟ(普閩典)。例:~,我明白了。(台)喃ㄏㄛㄕ 喔·ㄧ,我ㄍㄨㄚˋ 知ㄗㄞ(/知ㄗㄞ影ㄧㄚˋ)喇(/啦)ㄆㄌㄚㄟ。異文:噢ㄜ(陸美) ㄛˊ。反切見【:】音韻 國音ㄛˋ。(另見)ㄛ、ㄛˇ:ㄛˇ、ㄏㄨㄛˋ。 【陽ㄛˇ】[情]表:廟應答。例:~,是噢:ㄍㄢ!(陵美)。(台)哈ㄏㄚㄕ /ㄏㄚㄟ啊ㄇㄟ)有ㄨㄕ影ㄧㄚˋ按ㄋㄟ爾·ㄋㄧ頭 ㄏㄛㄕ 。 類語:約ㄟˋ。美ㄓˋ。(/暗)ㄩ。品(/圈)反,腳陽。嗢 ㄛ[·o] 喔 豆。 國音。 ㄧ。(台甘)、(普閩)的反切——互見【陽

FIGURE 5.3 – Comparaison entre référence et sortie (corpus50_T)

Cette visualisation des résultats nous permet de mieux nous rendre compte de

la qualité des résultats. Nous pouvons voir qu'avec le modèle corpus50_T, une post-édition manuelle n'est pas à proscrire.

5.8 Conclusion

Nous avons essayé plusieurs combinaisons de données pour améliorer les résultats. Cependant elles n'ont pas permis d'amélioration. Les résultats sont plutôt proches entre eux pour la f-mesure. Seul le CER a des différences plus marquantes entre les corpus. Certaines combinaisons ont même fait dégrader la reconnaissance.

CHAPITRE 6

DISCUSSION

6.1 Introduction

Dans ce dernier chapitre, nous allons faire un point sur les travaux effectués et les améliorations possibles.

6.2 Travaux effectuées

6.2.1 Les outils

eScriptorium

Le choix de travaillé avec eScriptorium a été une bonne chose. Le segmenteur est très performant. Un ré-entraînement avec peu de données a parmi une amélioration considérable. Même si le nouveau segmenteur entrainé est arrivé à un niveau très satisfaisant, la segmentation que nous avons choisi n'est peut-être pas la plus efficace. La pagination aurait pu être ignoré mais nous avons voulu rester fidèle au support physique pour inclure cette information dans la TEI même si finalement elle n'est pas utilisée. Par contre l'interface buggue parfois. Il est arrivé qu'il ne montre plus toutes les informations de segmentation tel que les zones de texte et la *baseline*.

kraken

Kraken permet le *fine-tuning* (continuer l'entraînement) d'un modèle avec de nouvelles données. Ce qui rend possible un gain de temps non négligeable. Cependant, dans notre cas, les nouvelles données ajoutées dégradaient le résultat du modèle existant. Pour l'apprentissage d'un modèle, il fallait à chaque fois recommencer l'entraînement avec tous les corpus que l'on souhaite intégré au modèle et cela prenait au moins trois jours. On peut évaluer un modèle avec kraken mais il ne donne pas toutes les informations pour qu'on puisse faire les calculs d'évaluation. Les mesures de rappel, précision et f-mesure nécessite le nombre de caractères de la sortie et de la référence. En utilisant l'évaluation de kraken, nous n'avons que le nombre total de caractères de la référence.

KaMI

KaMI est un bon outil récent qui nous donne le choix de la manière dont on veut l'utiliser. Il est possible de l'utiliser par son interface web ou en installant son API en python. Il permet d'évaluer des transcriptions pour la reconnaissance de texte

manuscrite et la reconnaissance optique de caractères à travers plusieurs mesures. Il contient les mesures CER, distance de leveinshtein et d'autres mais ne donne pas le rappel, la précision et la f-mesure. Son interface permet d'avoir un visuel pour comparer le texte de référence et celui de la sortie pour mieux nous rendre compte de ce que signifient les résultats qu'on obtient.

6.3 Travaux futurs

6.3.1 Modèles

Fusion de deux modèles

Comme nous l'avons abordé dans les résultats du corpus_TZhuyin, nous pourrions créer deux modèles spécifiques. Un modèle pour le *zhuyin fuhao* et un deuxième pour les sinogrammes. Pour chaque modèle, les caractères qui ne nous intéressent pas sont remplacés par un point. Nous pouvons ensuite fusionner les sorties des deux modèles en inter-changeant les points par les caractères.

Décomposition de caractères

Le dictionnaire étant d'origine taiwanaise, il est écrit dans les système d'écriture traditionnel. Les sinogrammes qui y figurent sont donc dans l'écriture "compliquée" qui peut comporter énormément de traits. Certains caractères ne sont pas dans Unicode et ne possède pas de police pouvant les afficher. Il serait peut-être possible d'essayer de créer un modèle de décomposition des caractères à l'aide du projet CHISE (*CHaracter Information Service Environment*)[1] pour qu'on puisse les réassembler après. Dans les données de la TEI, certains caractères sont déjà décomposé car leur glyphe n'existe pour l'instant pas.

6.3.2 Post-édition

Création d'un dictionnaire

Dans les erreurs de reconnaissance des caractères, certains pourraient être corrigés avec un dictionnaire car ce sont des caractères qui mis côte à côte n'ont pas de sens.

Crowdsourcing

Le crowdsourcing pour la post-édition serait une option. Nous avons déjà une interface disponible avec eScriptorium permettant de visualiser les lignes et de corriger manuellement les sorties. Il faudrait alors mettre en place une ressource pour la décomposition des caractères à porté de main car il est certain que des caractères ne pourront pas être tapés.

6.4 Conclusion

Nous avons vu que les outils utilisés sont performants mais ont quelques défauts. Que ce soit pour l'interface graphique eScriptorium ou l'apprentissage des

1. https://www.chise.org/

modèles avec kraken. Cependant, il n'est pas trop compliqué de les prendre en main avec les tutoriels qu'ils mettent à disposition. Il reste de nombreuses possibilités d'amélioration des résultats par des approches différentes dans l'apprentissage des modèles ou par la post-édition.

Conclusion générale

Dans ce mémoire, nous avons traité de la problématique de la reconnaissance optique de caractères d'un dictionnaires en mandarin-taiwanais. Son objectif est la récupération des données manquantes du dictionnaire par un modèle spécifique. Compte tenu des résultats que nous avons obtenus, nous pouvons dire que le couteau suisse qu'est eScriptorium basé kraken nous a montré des résultats satisfaisants. Les facteurs qui influent sur les résultats sont de l'ordre quantitatif et qualitatif. D'une certaine manière, nous somme limité par la quantité des données disponibles. Qualitatif car les données c'est-à-dire les images que nous disposions ne nécessitaient pas de pré-traitement. La seule mesure qui a vraiment montré une différence marquante entre les modèles est le CER.

BIBLIOGRAPHIE

[AdobeOriginals, nd] AdobeOriginals (n.d.). Ocr-b. – Cité page 23.

[Ait-Mohand et al., 2010] Ait-Mohand, K., Heutte, L., Paquet, T., and Ragot, N. (2010). Font adaptation of an hmm-based ocr system. In *Document Recognition and Retrieval XVII*, volume 7534, pages 173–180. SPIE. – Cité page 24.

[Barque et al., 2010] Barque, L., Nasr, A., and Polguere, A. (2010). From the definitions of the" trésor de la langue française" to a semantic database of the french language. In *XIV Euralex International Congress*, pages 245–252. – Cité page 17.

[Beagrie, 2003] Beagrie, N. (2003). National digital preservation initiatives: An overview of developments in australia, france, the netherlands, and the united kingdom and of related international activity. – Cité page 16.

[Belaïd and Cecotti, 2006] Belaïd, A. and Cecotti, H. (2006). Reconnaissance de caractères: évaluation des performances. – Cité page 28.

[Bernard, 2010] Bernard, P. (2010). Le trésor de la langue française informatisé. *Traduire. Revue française de la traduction*, (222):125–136. – Cité page 17.

[Bieniecki et al., 2007] Bieniecki, W., Grabowski, S., and Rozenberg, W. (2007). Image preprocessing for improving ocr accuracy. In *2007 International Conference on Perspective Technologies and Methods in MEMS Design*, pages 75–80. – Cité page 25.

[Breuel, 2008] Breuel, T. M. (2008). The ocropus open source ocr system. In *Document recognition and retrieval XV*, volume 6815, pages 120–134. SPIE. – Cité page 33.

[Casey and Nagy, 1966] Casey, R. and Nagy, G. (1966). Recognition of printed chinese characters. *IEEE Transactions on Electronic Computers*, EC-15(1):91–101. – Cité page 27.

[Deegan and Tanner, 2004] Deegan, M. and Tanner, S. (2004). Conversion of primary sources. *A companion to digital humanities*, pages 488–504. – Cité page 16.

[Dmitriy Genzel, 2015] Dmitriy Genzel, Ashok Popat, h. N. (2015). Paper to digital in 200+ languages. – Cité page 24.

[Dongre and Mankar, 2011] Dongre, V. J. and Mankar, V. H. (2011). A review of research on devnagari character recognition. *arXiv preprint arXiv:1101.2491*. – Cité page 26.

[Farhat et al., 2016] Farhat, A., Al-Zawqari, A., Al-Qahtani, A., Hommos, O., Bensaali, F., Amira, A., and Zhai, X. (2016). Ocr based feature extraction and template matching algorithms for qatari number plate. In *2016 International Conference on Industrial Informatics and Computer Systems (CIICS)*, pages 1–5. IEEE. – Cité page 24.

[Feng et al., 2019] Feng, X., Yao, H., and Zhang, S. (2019). Focal ctc loss for chinese optical character recognition on unbalanced datasets. *Complexity*, 2019. – Cité page 28.

[Gowan, 1995] Gowan, W. A. (1995). Optical character recognition using fuzzy logic. *Microprocessors and Microsystems*, 19(7):423–434. – Cité page 26.

[Hamad and Mehmet, 2016] Hamad, K. and Mehmet, K. (2016). A detailed analysis of optical character recognition technology. *International Journal of Applied Mathematics Electronics and Computers*, (Special Issue-1):244–249. – Cité pages 24 et 25.

[Hasnat et al., 2007] Hasnat, M., Habib, S., Khan, M., et al. (2007). Segmentation free bangla ocr using hmm: Training and recognition. – Cité page 24.

[Heylen and Kempf, 2001] Heylen, A. and Kempf, D. (2001). De l'histoire locale à l'histoire nationale [la difficile institutionnalisation d'une historiographie taiwanaise]. *Perspectives chinoises*, 66(1):41–54. – Cité page 18.

[Ide and Véronis, 1996] Ide, N. and Véronis, J. (1996). Codage tei des dictionnaires électroniques. *Cahiers GUTenberg*, (24):170–176. – Cité page 42.

[Jovian and Amprimo, 2011] Jovian, L. T. and Amprimo, O. (2011). Ocr correction via human computational game. In *2011 44th Hawaii International Conference on System Sciences*, pages 1–10. IEEE. – Cité page 26.

[Karpinski and Belaid, 2016] Karpinski, R. and Belaid, A. (2016). Rapport Evaluation des OCR. Research report, LORIA - Université de Lorraine. – Cité pages 28 et 29.

[Kiessling, 2019] Kiessling, B. (2019). Kraken - a Universal Text Recognizer for the Humanities. – Cité page 36.

[Klöter, 2005] Klöter, H. (2005). *Written taiwanese*, volume 2. Otto Harrassowitz Verlag. – Cité page 18.

[LeCun et al., 1995] LeCun, Y., Bengio, Y., et al. (1995). Convolutional networks for images, speech, and time series. *The handbook of brain theory and neural networks*, 3361(10):1995. – Cité page 23.

[Mori et al., 1992] Mori, S., Suen, C., and Yamamoto, K. (1992). Historical review of ocr research and development. *Proceedings of the IEEE*, 80(7):1029–1058. – Cité page 22.

[Patel et al., 2012] Patel, C., Patel, A., and Patel, D. (2012). Optical character recognition by open source ocr tool tesseract: A case study. *International Journal of Computer Applications*, 55(10):50–56. – Cité page 24.

[Patil et al., 2015] Patil, V., Sanap, R. V., and Kharate, R. B. (2015). Optical character recognition using artificial neural network. *Int. J. Eng. Res. Gen. Sci*, 3(1):7. – Cité page 22.

[Pierrel et al., 2004] Pierrel, J.-M., Dendien, J., and Bernard, P. (2004). Le tlfi ou trésor de la langue française informatisé. *Actes de EURALEX*, 4. – Cité page 17.

[PlanèteTypographie, nd] PlanèteTypographie (n.d.). Ocr-a. – Cité page 23.

[Rao et al., 2016] Rao, N. V., Sastry, A., Chakravarthy, A., and Kalyanchakravarthi, P. (2016). Optical character recognition technique algorithms. *Journal of Theoretical & Applied Information Technology*, 83(2). – Cité pages 22 et 26.

[Sabourin and Mitiche, 1992] Sabourin, M. and Mitiche, A. (1992). Optical character recognition by a neural network. *Neural Networks*, 5(5):843–852. – Cité page 23.

[Sharma et al., 2013] Sharma, O. P., Ghose, M., Shah, K. B., and Thakur, B. K. (2013). Recent trends and tools for feature extraction in ocr technology. *International Journal of Soft Computing and Engineering*, 2(6):220–223. – Cité page 27.

[Shinde and Chougule, 2012] Shinde, A. A. and Chougule, D. (2012). Text preprocessing and text segmentation for ocr. *International Journal of Computer Science Engineering and Technology*, 2(1):810–812. – Cité page 26.

[Smith, 2007] Smith, R. (2007). An overview of the tesseract ocr engine. In *Ninth international conference on document analysis and recognition (ICDAR 2007)*, volume 2, pages 629–633. IEEE. – Cité page 23.

[Stallings, 1972] Stallings, W. (1972). Recognition of printed chinese characters by automatic pattern analysis. *Computer Graphics and Image Processing*, 1(1):47–65. – Cité page 27.

[Strankale and Paikens, 2020] Strankale, L. and Paikens, P. (2020). Ocr challenges for a latvian pronunciation dictionary. In *Baltic HLT*, pages 199–206. – Cité page 24.

[Terras, 2011] Terras, M. M. (2011). *The Rise of Digitization*, pages 3–20. SensePublishers, Rotterdam. – Cité page 15.

[Verma and Ali, 2012] Verma, R. and Ali, J. (2012). A-survey of feature extraction and classification techniques in ocr systems. *International Journal of Computer Applications & Information Technology*, 1(3):1–3. – Cité page 27.

[Wolf et al., 2001] Wolf, C., Jolion, J.-M., and Chassaing, F. (2001). Détection et extraction de texte de la vidéo. *7èmes Journées CORESA*, 1:251–258. – Cité page 23.

[Zhang et al., 2013] Zhang, J., Wu, X., Yu, Y., and Luo, D. (2013). A method of neighbor classes based svm classification for optical printed chinese character recognition. *Plos one*, 8(3):e57928. – Cité page 27.

[Zhang et al., 2018] Zhang, J., Zhu, Y., Du, J., and Dai, L. (2018). Radical analysis network for zero-shot learning in printed chinese character recognition. In *2018 IEEE International Conference on Multimedia and Expo (ICME)*, pages 1–6. IEEE. – Cité page 27.

[Zhong et al., 2015] Zhong, Z., Jin, L., and Feng, Z. (2015). Multi-font printed chinese character recognition using multi-pooling convolutional neural network. In *2015 13th International Conference on Document Analysis and Recognition (ICDAR)*, pages 96–100. IEEE. – Cité page 27.

[台灣維基媒體協會 , 2018] 台灣維基媒體協會 (2018). . – Cité page 18.

[吳守禮 , 2018] 吳守禮 (2018). xml-tei. – Cité page 44.

ANNEXE A

LES SCRIPTS

A.1 Script pour l'extraction des données de la TEI

A.2 Script pour la génération des images

A.3 Script pour la création du fichier page

A.4 Script pour les données du fichier page

```
noteTypes = ['國音', '台甘', '普閩']

# pour chaque grand chapitre syllabique
for div in Tei.select('div[n]'):
    for head in div.select('head'):
        print(head.text)
    for p in div.select('p'):
        print(zhuyinCara(p))
    for divS in div.select('div [type$="sinograms"]'):
        # print(divS)
        for entryFree in divS.select('entryFree'):
            # print(entryFree)
            form = entryFree.select_one('form')
            # print(form)
            for orth in form.select('orth'):
                o = orth.text
            for g in form.select('pron g'):
                # print(g)
                m = re.match(r'\<g ref="(to-check|ruby|mapped|hj|missing)-(F|f)(M3|m3|K|k)-f(\w{4})">.+\</g>', str(g))
                m2 = m2 = re.findall(r'\<rt>(.)\|F(M3|K)\|\w{5}\</rt>', str(g))
                m3 = re.findall(r'\<g ref="ruby--0">\</g>', str(g))
                if m is None:
                    cont = o
                elif m:
                    notag = re.sub(r'\<g ref="(to-check|ruby|mapped|hj|missing)-(F|f)(M3|m3|K|k)-f(\w{4})">.+\</g>', r'-\3/\4.png|', str(g)).lower()
                    cont = o + "|" + str(g.text) + notag
                elif m2:
                    notag2 = re.sub(r'\<rt>(.)\|F(M3|K)\|f\w{4}\</rt>', r'\1', cont)
                    cont = notag2
                elif m3:
                    notag3 = re.sub(r'\<g ref="ruby--0">\</g>', '', str(g))
                    cont = notag3
            # différentes prononciations
            for g1 in form.select('note[type="comment"]'):
                print(cont + zhuyinCara(g1))
                note = ""
                for nt in noteTypes:
                    for n in form.select('note[type="' + nt + '"]'):
                        note = note + zhuyinCara(n, nt)
                print(note)
            for superEntry in entryFree.select('superEntry'):
                # print(superEntry)
                # print(extractEntry(superEntry))
                for entry in superEntry.select('entry'):
                    # print(entry)
                    for e in entry.select('orth[type="full"]'):
                        # print(e)
                        defpre = zhuyinCara(e)
                    # on sélectionne la définition qui va avec
                    for de in entry.select('def'):
                        definition = defpre + zhuyinCara(de)
                    print(definition)

    for divW in div.select('div [type$="words"]'):
        # print(divW)
        for entry in divW.select('entry'):
            for e in entry.select('orth[type="full"]'):
                # print(e)
                defpre = zhuyinCara(e)
            # on sélectionne la définition qui va avec
            for de in entry.select('def'):
                definition = defpre + zhuyinCara(de)
            print(definition)
```

FIGURE A.1 – Extrait du script d'extraction

```
# ------------------------------------------------------------------------------------------------------------------------

# pour chaque ligne du dico
for ligne in dico_tei:
    ligne = ligne.strip('\n')
    # recherche s'il y a des chemins de de zhuyin
    ligne = re.findall(r'(?:\|(?P<img>(?:.+?)-((?:m3|k)/\w{4}.png))\|)|(?P<txt>[^\|]+)', ligne)
    # print(len(ligne), ligne, type(ligne))
    d, img, text, curseur, font, tailleImage = newImg(WIDTH, HEIGHT)
    # taille de l'image caractère voulue
    nsizeW, nsizeH = (tailleImage, tailleImage)
    placeImage = int((HEIGHT-nsizeH)/2)

    if len(ligne) < 2:
        for group in ligne:
            tailleFontT = random.randint(70, 80)
            f1 = ImageFont.truetype('{}arphic/uming.ttc'.format(fontt), tailleFontT, encoding="unic")
            f2 = ImageFont.truetype('{}arphic/ukai.ttc'.format(fontt), tailleFontT, encoding="unic")
            f3 = ImageFont.truetype('{}noto/NotoSerifCJK-Regular.ttc'.format(fonto), tailleFontT, encoding="unic")
            fontT = random.choice((f1, f2, f3))
            d, curseur, text, sizeH, lstCara = generCara(group[2], text, fontT, d, curseur, HEIGHT, nsizeW, lstCara)

            i = sortieFI(chemin, i, sizeH, WIDTH, HEIGHT, curseur, text)
    else:
        for n,group in enumerate(ligne):
            if (curseur + nsizeW) < (WIDTH - 30):
                zhuyin = group[0].split('-')
                if group[1]:
                    path = '../koktai/img/{}'.format(group[1])
                    # on vérifie que le chemin existe
                    if os.path.exists(path):
                        # on ouvre l'image déjà créé par un caractère normale
                        img, curseur = casImg(path, nsizeW, nsizeH, img, curseur, placeImage)

                        dictioImg = dictImg(dictioImg, zhuyin[0], path)

                    else:
                        sizeW, sizeH = d.textsize(zhuyin[0], font=font)
                        if curseur + (sizeW or nsizeW) < (WIDTH - 30):
                            # insertion du texte dans l'image
                            d, curseur = casCara(zhuyin[0], font, d, curseur, HEIGHT)
                            lstCara = listeCara(lstCara, zhuyin[0])
                # concaténation du texte de base avec le zhuyin
                text = text + zhuyin[0]
                # print(text, "dans 2.1")
                if group[2]:
                    countCara = 0
                    for caractere in group[2]:
                        sizeW, sizeH = d.textsize(caractere, font=font)
                        countCara += 1
                        if curseur + (sizeW or nsizeW) < (WIDTH - 30):
                            # insertion du texte dans l'image
                            d, curseur = casCara(caractere, font, d, curseur, HEIGHT)
                            lstCara = listeCara(lstCara, caractere)
                            # concaténation des caractères
                            text = text + caractere

                        if n == len(ligne)-1 and countCara == len(group[2]):
                            i = sortieFI(chemin, i, sizeH, WIDTH, HEIGHT, curseur, text)

                            # création d'une nouvelle image
                            d, img, text, curseur, font, tailleImage = newImg(WIDTH, HEIGHT)
                    else:
                        i = sortieFI(chemin, i, sizeH, WIDTH, HEIGHT, curseur, text)

                        # création d'une nouvelle image
                        d, img, text, curseur, font, tailleImage = newImg(WIDTH, HEIGHT)

                        d, curseur = casCara(caractere, font, d, curseur, HEIGHT)
                        lstCara = listeCara(lstCara, caractere)
                        text = text + caractere
            else:
                i = sortieFI(chemin, i, sizeH, WIDTH, HEIGHT, curseur, text)

                # création d'une nouvelle image
                d, img, text, curseur, font, tailleImage = newImg(WIDTH, HEIGHT)

                if group[1]:
                    zhuyin = group[0].split('-')
                    path = '../koktai/img/{}'.format(group[1])

                    # concaténation du texte de base avec le zhuyin
                    text = text + zhuyin[0]

                    # on vérifie que le chemin existe
                    if os.path.exists(path):
                        # on ouvre l'image déjà créé par un caractère normale
                        img, curseur = casImg(path, nsizeW, nsizeH, img, curseur, placeImage)

                        dictioImg = dictImg(dictioImg, zhuyin[0], path)
                    else:
                        sizeW, sizeH = d.textsize(zhuyin[0], font=font)
                        if curseur + (sizeW or nsizeW) < (WIDTH - 30):
                            # insertion du texte dans l'image
                            d, curseur = casCara(zhuyin[0], font, d, curseur, HEIGHT)
                            lstCara = listeCara(lstCara, zhuyin[0])
                if group[2]:
                    d, curseur, text, sizeH, lstCara = generCara(group[2], text, font, d, curseur, HEIGHT, nsizeW, lstCara)


with open ("../images/soloCaractere/listCaractere.txt", "w") as listeCara:
    json.dump(lstCara, listeCara)
    print("Liste de caractères créés")
```

FIGURE A.2 – Extrait du script GenerImg.py

```python
def creaXML(date, nomFI, WIDTH, HEIGHT, zoneT, mask, baseline, transcription, fichier):

    attr_qname = ET.QName("http://www.w3.org/2001/XMLSchema-instance", "schemaLocation")

    nsmap = {None: "http://schema.primaresearch.org/PAGE/gts/pagecontent/2019-07-15",
             "xsi": "http://www.w3.org/2001/XMLSchema-instance"
             }

    root = ET.Element("PcGts", {attr_qname:"http://schema.primaresearch.org/PAGE/gts/pagecontent/2019-07-15 http://schema.primaresearch.org/PAGE/gts/pagecontent/2019-07-15/pagecontent.xsd"}, nsmap=nsmap)

    Metadata = ET.SubElement(root, "Metadata")
    Creator =  ET.SubElement(Metadata, "Creator")
    Creator.text = "Afala PHAXAY"
    Created = ET.SubElement(Metadata, "Created")
    Created.text = str(date)
    LastChange = ET.SubElement(Metadata, "LastChange")
    LastChange.text = str(date)

    Page = ET.SubElement(root, "Page")
    Page.set("imageFilename", nomFI)
    Page.set("imageWidth", str(WIDTH))
    Page.set("imageHeight", str(HEIGHT))

    TextRegion = ET.SubElement(Page, "TextRegion")
    TextRegion.set("id", "block_0")
    TextRegion.set("custom", "structure {type:text;}")

    Coords1 = ET.SubElement(TextRegion, "Coords")
    Coords1.set("points", zoneT)

    TextLine = ET.SubElement(TextRegion, "TextLine")
    TextLine.set("id", "line_0")

    Coords2 = ET.SubElement(TextLine, "Coords")
    Coords2.set("points", mask)

    Baseline = ET.SubElement(TextLine, "Baseline")
    Baseline.set("points", baseline)

    TextEquiv = ET.SubElement(TextLine, "TextEquiv")

    Unicode = ET.SubElement(TextEquiv, "Unicode")
    Unicode.text = transcription

    tree = ET.ElementTree(root)
    tree.write(fichier, pretty_print=True, encoding='utf-8', xml_declaration=True)
```

FIGURE A.3 – Sortie PAGEXML

```
def coordonnees(curseur, sizeH):

    zoneT = list()
    mask = list()
    baseline = list()

    marge = int((HEIGHT-sizeH)/2)
    marge1 = int(0.2*sizeH)
    marge2 = int(0.1*sizeH)

    b1 = 45, marge - marge2
    b2 = b1[0], marge + sizeH + marge1
    b3 = curseur + int(sizeH*0.2), b2[1]
    b4 = b3[0], b1[1]

    zoneT.append(b1)
    zoneT.append(b2)
    zoneT.append(b3)
    zoneT.append(b4)

    baselineM =  marge + sizeH

    l1o = random.randint(marge + marge1, baselineM - marge2)
    l2e = random.randint(marge + marge1, baselineM - marge2)

    l1 = 45, l1o
    l2 = curseur + int(sizeH*0.1), l2e

    baseline.append(l1)
    baseline.append(l2)

    # coordonnées du mask polygon
    m1 = 50, b1[1]
    m2 = curseur, m1[1]
    m3 = m2[0], b2[1]
    m4 = m1[0], b3[1]

    mask.append(m1)
    mask.append(m2)
    mask.append(m3)
    mask.append(m4)


    zoneT = ' '.join('{},{}'.format(*coord) for coord in zoneT)

    mask = ' '.join('{},{}'.format(*coord3) for coord3 in mask)

    baseline = ' '.join('{},{}'.format(*coord2) for coord2 in baseline)

    return zoneT, mask,baseline
```

FIGURE A.4 – Données de structure

ANNEXE B

CORPUS ET MODÈLES

```
<div>
    <head></head>
    <p/>
    <div type="sinograms">
        <entryFree type="sinogram">
            <form>
                <orth></orth>
                <pron><g></g></pron>
                <note type="comment"></note>
                <note type="國音"><g></g></note>
                <note type="台甘"><g></g></note>
                <note type="普閩"><g></g></note>
            </form>
            <superEntry>
                <entry type="word">
                    <form>
                        <orth type="full"
                        ><g></g></orth>
                        <orth type="no-zhuyin"
                        ></orth>
                        <pron></pron>
                    </form>
                    <gramGrp>
                        <po></po>
                    </gramGrp>
                    <def><g ref=""></g></def>
                </entry>
            </superEntry>
        </entryFree>
    </div>
</div>
```

```
    </div>
    <div>
        <head></head>
        <p/>
        <div type="words">
            <entry type="word">
                <form>
                    <orth type="full"><g></g></orth>
                    <orth type="no-zhuyin"></orth>
                    <pron></pron>
                </form>
                <gramGrp>
                    <pos></pos>
                </gramGrp>
                <def><g ref=""></g></def>
            </entry>
            <entry type="word">
                <form>
                    <orth type="full"><g></g></orth>
                    <orth type="no-zhuyin"></orth>
                    <pron></pron>
                </form>
                <gramGrp>
                    <pos></pos>
                </gramGrp>
                <def><g ref=""></g></def>
            </entry>
        </div>
</body>
```

FIGURE B.1 – Structure de la TEI

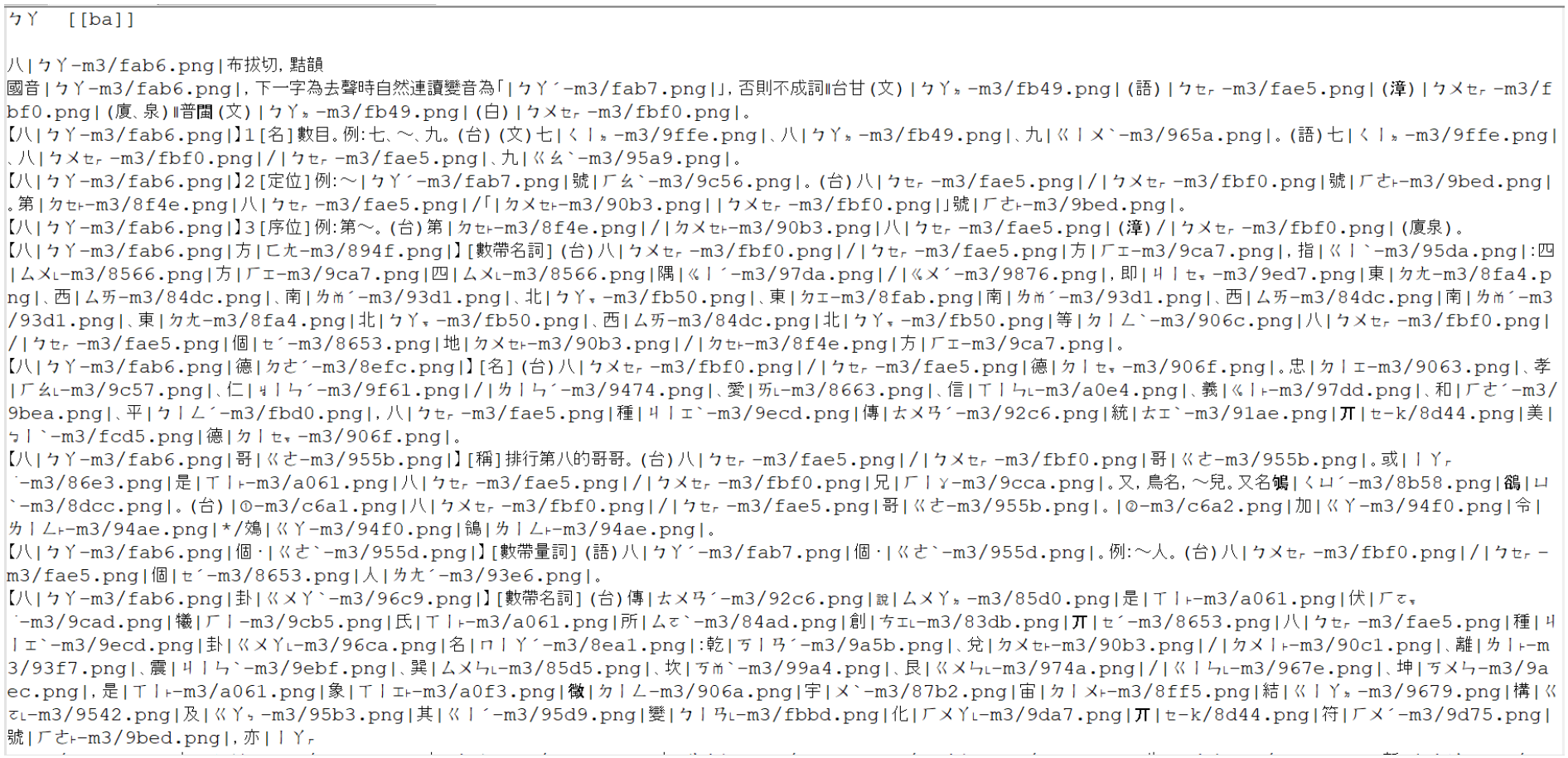
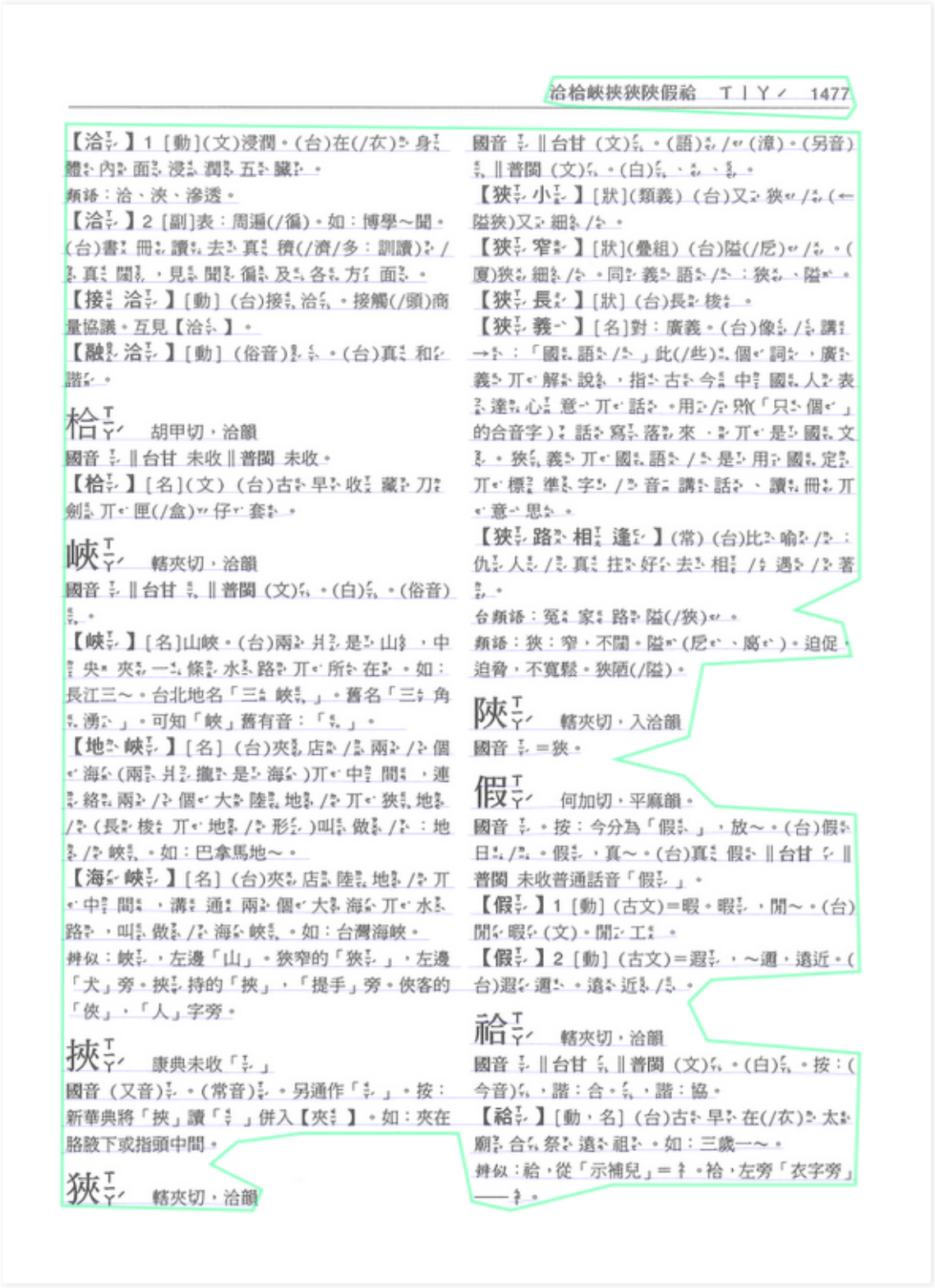


FIGURE B.2 – Sortie de l'extraction



洽柙峽挾狹陝假袷 ㄒㄧㄚˊ 1477

【洽ㄒㄧㄚ】1［動］(文)浸潤。(台)在(/衣)ㄉ身ㄊ體ㄊˇ內ㄋˋ面ㄇˋ浸ㄐㄧㄣˋ潤ㄖㄨㄣˋ五ㄨˇ臟ㄗㄤˋ。
頻語：洽、浹、滲透。
【洽ㄒㄧㄚ】2［副］表：周遍(/徧)。如：博學～聞。(台)書ㄕ冊ㄘㄜˋ讀ㄉㄨˊ去ㄑㄩˋ真ㄓㄣ積(/濟/多：訓讀)ㄉㄜ/ㄉㄜˊ真ㄓㄣ閩ㄇㄧㄣˊ，見ㄐㄧㄢˋ聞ㄨㄣˊ徧ㄅㄧㄢˋ及ㄐㄧˊ各ㄍㄜˋ方ㄈㄤ面ㄇㄧㄢˋ。
【接ㄐㄧㄝ洽ㄒㄧㄚ】［動］(台)接ㄐㄧㄝ洽ㄑㄧㄚˋ。接觸(/頭)商量協議。互見【洽ㄑㄧㄚˋ】。
【融ㄖㄨㄥˊ洽ㄒㄧㄚ】［動］(俗音)ㄏㄜˊ。(台)真ㄓㄣ和ㄏㄜˊ諧ㄒㄧㄝˊ。

柙 ㄒㄧㄚˊ 胡甲切，洽韻
國音ㄒㄧㄚˊ‖台甘 未收‖普閩 未收。
【柙ㄒㄧㄚˊ】［名］(文)(台)古ㄍㄨˇ早ㄗㄠˇ收ㄕㄡ藏ㄘㄤˊ刀ㄉㄠ劍ㄐㄧㄢˋ丌ㄑㄧˊ匣(/盒)ㄒㄧㄚˊ仔ㄗㄞˇ套ㄊㄠˋ。

峽 ㄒㄧㄚˊ 轄夾切，洽韻
國音ㄒㄧㄚˊ‖台甘ㄒㄧㄚˊ‖普閩 (文)ㄒㄧㄚˊ。(白)ㄍㄜˊ。(俗音)ㄏㄜˊ。
【峽ㄒㄧㄚˊ】［名］山峽。(台)兩ㄌㄧㄤˇ爿ㄆㄤˊ是ㄕˋ山ㄕㄢ，中ㄓㄨㄥ央ㄧㄤ夾ㄐㄧㄚˊ一ㄧˊ條ㄊㄧㄠˊ水ㄕㄨㄟˇ路ㄌㄨˋ丌ㄑㄧˊ所ㄙㄨㄛˇ在ㄗㄞˋ。如：長江三～。台北地名「三ㄙㄢ峽ㄒㄧㄚˊ」。舊名「三ㄙㄢ角ㄐㄧㄠˇ湧ㄩㄥˇ」。可知「峽」舊有音：「ㄍㄜˊ」。
【地ㄉㄧˋ峽ㄒㄧㄚˊ】［名］(台)夾ㄐㄧㄚˊ店ㄉㄧㄢˋ/ㄉㄧㄢˋ兩ㄌㄧㄤˇ/ㄌㄧㄤˇ個ㄍㄜˋ海ㄏㄞˇ(兩ㄌㄧㄤˇ爿ㄆㄤˊ攏ㄌㄨㄥˇ是ㄕˋ海ㄏㄞˇ)丌ㄑㄧˊ中ㄓㄨㄥ間ㄐㄧㄢ，連ㄌㄧㄢˊ絡ㄌㄨㄛˋ兩ㄌㄧㄤˇ/ㄌㄧㄤˇ個ㄍㄜˋ大ㄉㄚˋ陸ㄌㄨˋ地ㄉㄧˋ/ㄉㄧˋ丌ㄑㄧˊ狹ㄒㄧㄚˊ地ㄉㄧˋ/ㄉㄧˋ(長ㄔㄤˊ梭ㄙㄨㄛ丌ㄑㄧˊ地ㄉㄧˋ/ㄉㄧˋ形ㄒㄧㄥˊ)叫ㄐㄧㄠˋ做ㄗㄨㄛˋ/ㄗㄨㄛˋ：地ㄉㄧˋ/ㄉㄧˋ峽ㄒㄧㄚˊ。如：巴拿馬地～。
【海ㄏㄞˇ峽ㄒㄧㄚˊ】［名］(台)夾ㄐㄧㄚˊ店ㄉㄧㄢˋ陸ㄌㄨˋ地ㄉㄧˋ/ㄉㄧˋ丌ㄑㄧˊ中ㄓㄨㄥ間ㄐㄧㄢ，講ㄐㄧㄤˇ通ㄊㄨㄥ兩ㄌㄧㄤˇ個ㄍㄜˋ大ㄉㄚˋ海ㄏㄞˇ丌ㄑㄧˊ水ㄕㄨㄟˇ路ㄌㄨˋ，叫ㄐㄧㄠˋ做ㄗㄨㄛˋ/ㄗㄨㄛˋ海ㄏㄞˇ峽ㄒㄧㄚˊ。如：台灣海峽。
辨似：峽ㄒㄧㄚˊ，左邊「山」。狹窄的「狹ㄒㄧㄚˊ」，左邊「犬」旁。挾ㄒㄧㄚˊ持的「挾」，「提手」旁。俠客的「俠」，「人」字旁。

挾 ㄒㄧㄚˊ 康典未收「ㄒㄧㄚˊ」
國音(又音)ㄐㄧㄚˊ。(常音)ㄑㄧㄚˋ。另通作「ㄏㄜˊ」。按：新華典將「挾」讀「ㄒㄧㄝˊ」併入【夾ㄐㄧㄚˊ】。如：夾在胳肢下或指頭中間。

狹 ㄒㄧㄚˊ 轄夾切，洽韻
國音ㄒㄧㄚˊ‖台甘(文)ㄒㄧㄚˊ。(語)ㄏㄜˊ/ㄏㄨㄛˊ(漳)。(另音)ㄒㄧㄚˊ‖普閩(文)ㄒㄧㄚˊ。(白)ㄍㄜˊ、ㄏㄜˊ、ㄒㄧㄚˊ。
【狹ㄒㄧㄚˊ小ㄒㄧㄠˇ】［狀］(類義)(台)又ㄧㄡˋ狹ㄏㄨㄛˊ/ㄒㄧㄚˊ(←隘狹)又ㄧㄡˋ細ㄙㄧˋ/ㄒㄧˋ。
【狹ㄒㄧㄚˊ窄ㄓㄞˇ】［狀］(疊組)(台)隘(/厄)ㄜˋ/ㄜˋ。(廈)狹ㄏㄚˊ細ㄙㄜˋ/ㄒㄧˋ。同ㄊㄨㄥˊ義ㄧˋ語ㄩˇ/ㄕˋ：狹ㄏㄚˊ、隘ㄞˋ。
【狹ㄒㄧㄚˊ長ㄔㄤˊ】［狀］(台)長ㄔㄤˊ梭ㄙㄨㄛ。
【狹ㄒㄧㄚˊ義ㄧˋ】［名］對：廣義。(台)像ㄒㄧㄤˋ/ㄒㄧㄤˋ講ㄐㄧㄤˇ→ㄕㄨㄛ：「國ㄍㄨㄛˊ語ㄩˇ/ㄕˋ」此(/些)ㄒㄧㄝ個ㄍㄜˋ詞ㄘˊ，廣ㄍㄨㄤˇ義ㄧˋ丌ㄑㄧˊ解ㄐㄧㄝˇ說ㄕㄨㄛ，指ㄓˇ古ㄍㄨˇ今ㄐㄧㄣ中ㄓㄨㄥ國ㄍㄨㄛˊ人ㄖㄣˊ表ㄅㄧㄠˇ達ㄉㄚˊ心ㄒㄧㄣ意ㄧˋ丌ㄑㄧˊ話ㄏㄨㄚˋ。用ㄩㄥˋ/ㄩㄥˋ咏(「只ㄓˇ個ㄍㄜˋ」的合音字)ㄓㄜˋ話ㄏㄨㄚˋ寫ㄒㄧㄝˇ落ㄌㄨㄛˋ來，ㄓㄜˋ丌ㄑㄧˊ是ㄕˋ國ㄍㄨㄛˊ文ㄨㄣˊ。狹ㄒㄧㄚˊ義ㄧˋ丌ㄑㄧˊ國ㄍㄨㄛˊ語ㄩˇ/ㄕˋ是ㄕˋ用ㄩㄥˋ國ㄍㄨㄛˊ定ㄉㄧㄥˋ丌ㄑㄧˊ標ㄅㄧㄠ準ㄓㄨㄣˇ字ㄗˋ/ㄗˋ音ㄧㄣ講ㄐㄧㄤˇ話ㄏㄨㄚˋ、讀ㄉㄨˊ冊ㄘㄜˋ丌ㄑㄧˊ意ㄧˋ思ㄙ。
【狹ㄒㄧㄚˊ路ㄌㄨˋ相ㄒㄧㄤ逢ㄈㄥˊ】(常)(台)比ㄅㄧˇ喻ㄩˋ/ㄩˋ：仇ㄔㄡˊ人ㄖㄣˊ/ㄗㄞˋ真ㄓㄣ拄ㄓㄨˇ好ㄏㄠˇ去ㄑㄩˋ相ㄒㄧㄤ/ㄒㄧㄤ遇ㄩˋ/ㄩˋ著ㄓㄠˊ。
台頻語：冤ㄩㄢ家ㄐㄧㄚ路ㄌㄨˋ隘(/狹)ㄜˋ。
頻語：狹：窄，不闊。隘ㄞˋ(厄ㄜˋ、窩ㄨㄛ)。迫促，迫脅，不寬鬆。狹陋(/隘)。

陝 ㄒㄧㄚˊ 轄夾切，入洽韻
國音ㄒㄧㄚˊ＝狹。

假 ㄒㄧㄚˇ 何加切，平麻韻。
國音ㄐㄧㄚˇ。按：今分為「假ㄐㄧㄚˇ」，放～。(台)假ㄐㄧㄚˇ日ㄖˋ/ㄐㄧㄚˇ。假ㄐㄧㄚˇ，真～。(台)真ㄓㄣ假ㄐㄧㄚˇ‖台甘ㄍㄜˇ‖普閩 未收普通話音「假ㄐㄧㄚˇ」。
【假ㄐㄧㄚˇ】1［動］(古文)＝暇。暇ㄒㄧㄚˊ，閒～。(台)閒ㄒㄧㄢˊ暇ㄒㄧㄚˊ(文)。閒ㄒㄧㄢˊ工ㄍㄨㄥ。
【假ㄐㄧㄚˇ】2［動］(古文)＝遐ㄒㄧㄚˊ，～邇，遠近。(台)遐ㄒㄧㄚˊ邇ㄦˇ。遠ㄩㄢˇ近ㄐㄧㄣˋ/ㄐㄧㄣˋ。

袷 ㄒㄧㄚˊ 轄夾切，洽韻
國音ㄐㄧㄚˊ‖台甘ㄍㄜˊ‖普閩(文)ㄍㄜˊ。(白)ㄐㄧㄚˊ。按：(今音)ㄍㄜˊ，諧：合。ㄍㄜˊ，諧：協。
【袷ㄐㄧㄚˊ】［動、名］(台)古ㄍㄨˇ早ㄗㄠˇ在(/衣)ㄉ太ㄊㄞˋ廟ㄇㄧㄠˋ合ㄍㄜˊ祭ㄐㄧˋ遠ㄩㄢˇ祖ㄗㄨˇ。如：三歲一～。
辨似：袷，從「示補兒」＝礿。袷，左旁「衣字旁」——礿。

FIGURE B.3 – Segmentation par le modèle eScriptorium

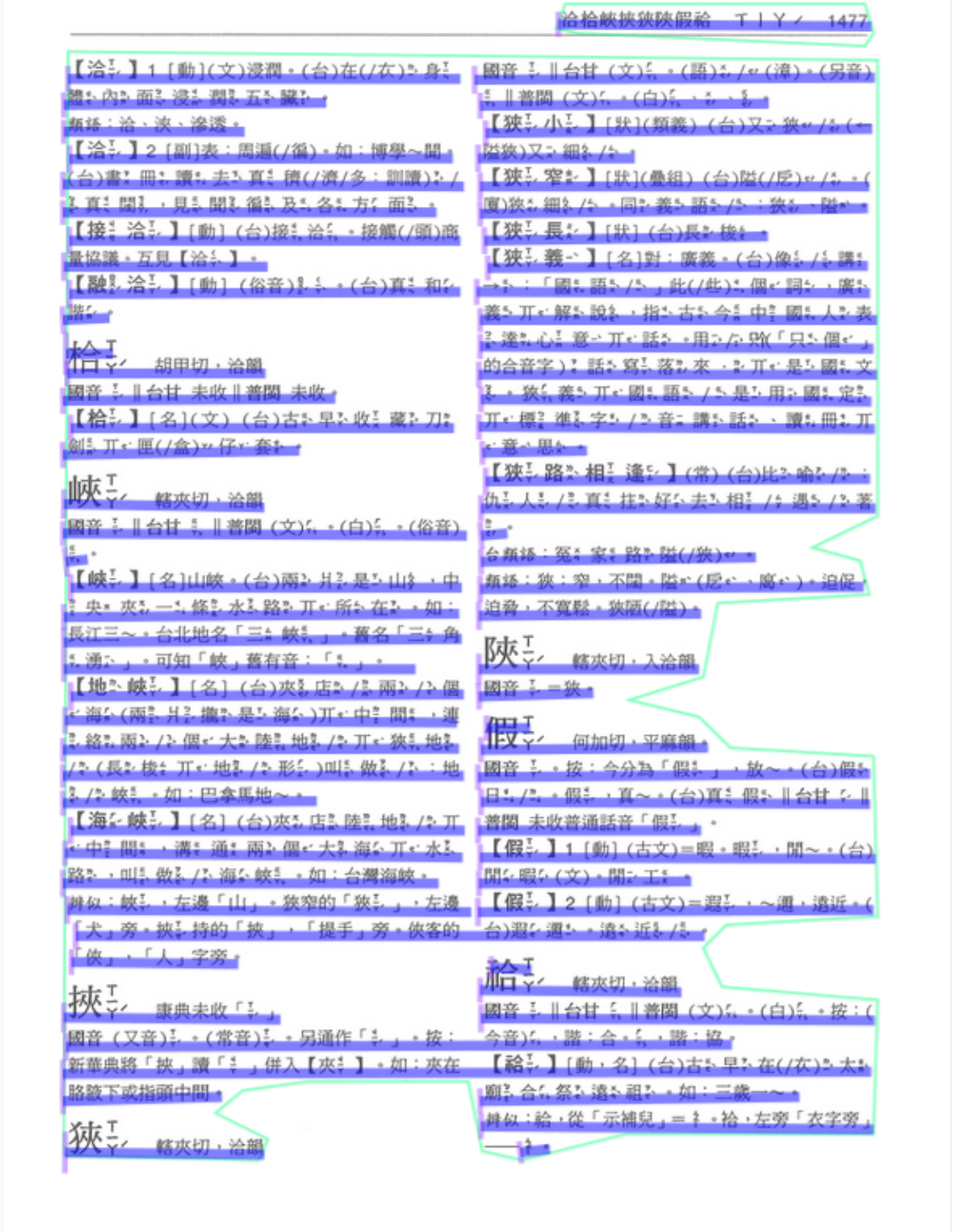


FIGURE B.4 – Reconnaissances des lignes de texte

FIGURE B.5 – Ordre des lignes

FIGURE B.6 – Fenêtre des manipulations

FIGURE B.7 – Zone de texte

FIGURE B.8 – Baseline

FIGURE B.9 – Ordre des lignes correcte dans les zones de texte

Index