Atelier OCR et HTR

Jean-Baptiste Camps

Masters HNC et TNAH | ENC (PSL)

8 janvier 2019

Acquisition du texte

Dans la constitution d'un corpus de textes, la première phase est bien sûr l'acquisition du contenu des textes envisagés.

Transcription des témoins, selon les critères scientifiques du projet (transcriptions allographétiques, graphématiques, normalisées ; édition critique ; etc.). Méthode souvent la plus sûre, mais aussi la plus lente ;

"Transcription" assistée par ordinateur en utilisant un algorithme permettant la reconnaissance optique de caractères (*optical character recognition* ou OCR) imprimés, ou la reconnaissance des écritures manuscrites (*handwritten text recognition* ou HTR).

Téléchargement de textes depuis des corpus en lignes, des sites d'édition électronique, des bases de données d'éditeurs, etc.

OCR et HTR

Reconnaissance optique des caractères (imprimés)
Optical character recognition (OCR)

- "problème résolu" de l'informatique ;
- aisé d'obtenir des taux d'erreur caractère (CER) $< 2\%$;
- outils libres : `Tesseract 4`, … ;
- existence de modèles génériques (par langue).

Reconnaissance des écritures manuscrites
Handwritten text recognition (HTR)

- très peu fonctionnel jusqu'à ces dernières années ;
- nouveaux développements : IA (réseaux de neurones récurrents LSTM…) ;
- outils libres : `Kraken`, … ;
- modèles spécifiques à entraîner (pour chaque main, écriture,…) ;
- on progresse petit à petit sur la généralité des modèles.

Les étapes

1. traitement des images ;
2. analyse de la mise en page et identification des lignes ;
3. reconnaissance des caractères ;
4. d'éventuels post-traitements, visant à améliorer les résultats.

Plan

Dans une démarche de reconnaissance des écritures, la qualité des images et de leurs traitements est cruciale.

Besoins :

1. images en 300 DPI minimum ;
2. redressées, débruitées ;
3. binarisées.

Outils : logiciels de traitement d'image, par ex. `ScanTailor`

T.P. ScanTailor

1. Démarrer un nouveau projet ;
2. charger les images du dossier `src` (Mer des Histoires ou Cid à défaut) ;
3. suivre les différentes étapes dans le logiciel ;
4. exporter en tiff binarisé 600 DPI.

Plan

Analyser la mise en page

Identifier

- zones de texte ;
- décoration ;
- colonnes ;
- lignes ;
- mots ;
- lettres.

Approches

- Sans apprentissage, par ex.
  - `OCRopy 1` ;
  - Oriflamms (IRHT) ;
  - `Kraken` ;
- fondée sur des méthodes d'apprentissage (IA), par ex.
  - `OCRopy 2` ?

Installer Kraken

Une installation de python ≥ 3.6 est nécessaire.

Sur Ubuntu, il vous faut les paquets `python3.6`, `python3.6-dev` et `pip3`.

```
# Créer un environnement virtuel (optionnel)
$ virtualenv env -p /usr/bin/python3.6
# l'activer (optionnel)
$ source env/bin/activate
# installer
$ pip3 install kraken
# Utilisation basique
# installer le modèle par défaut
$ kraken get default
# lister les modèles disponibles au téléchargement
$ kraken list
```

Analyser la mise en page avec Kraken : étape par étape

Depuis la racine du dossier :

Si les images n'ont pas été prétraitées

```
#Binariser les images (si pas déjà fait avec ScanTailor)
$ kraken -I "src/*" -o .png binarize
# Segmentation en lignes
$ kraken -I "src/*.png" -o .json segment
# OU les deux d'un coup
kraken -I "src/*" -o .json binarize segment
```

Si elles l'ont été

```
# Segmentation en lignes
$ kraken -I "tif/*" -o .json segment
```

Tout-en-un

Pour binariser, segmenter et générer un fichier de transcription

```
# Pour générer un fichier de transcription
$ ketos transcribe -o gt.html tif/*
```

Plan

Différentes solutions techniques

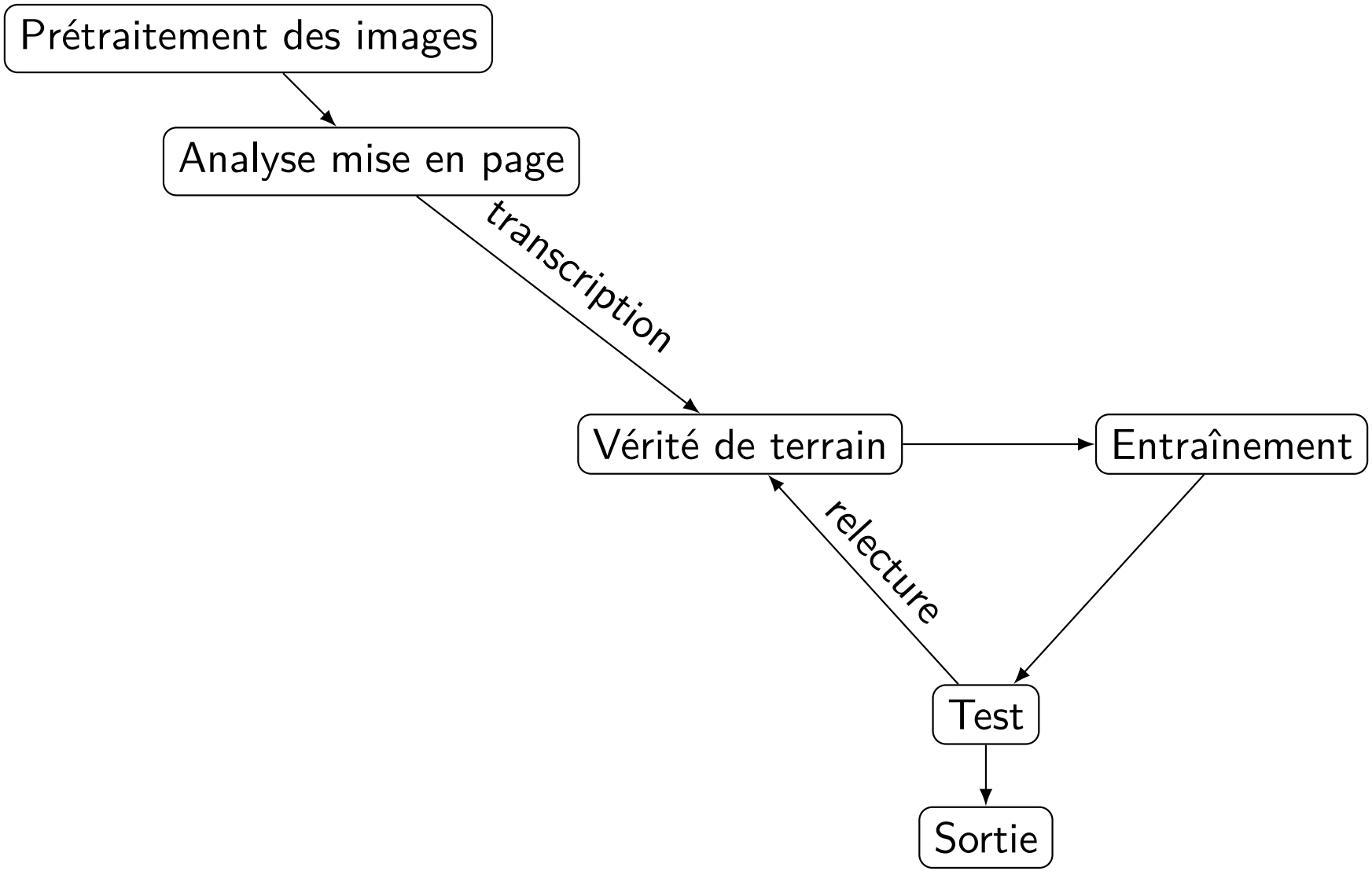
- approches segmentées ou non segmentées ;
- mesures de distance ; méthodes statistiques ou d'intelligence artificielle (réseaux de neurones convolutifs ou récurrents, LSTM 1D, LSTM 2D, etc.) ;
- outils directement opérationnels ou nécessitant un entraînement.

Ocropy, CLSTM et Kraken

`OCRopy` et CLSTM développés par Thomas M. Breuel ; `Kraken`, *fork* d'`OCRopy` développé par Ben Kiessling (PSL), `Calamari` (*fork* de Kraken)…

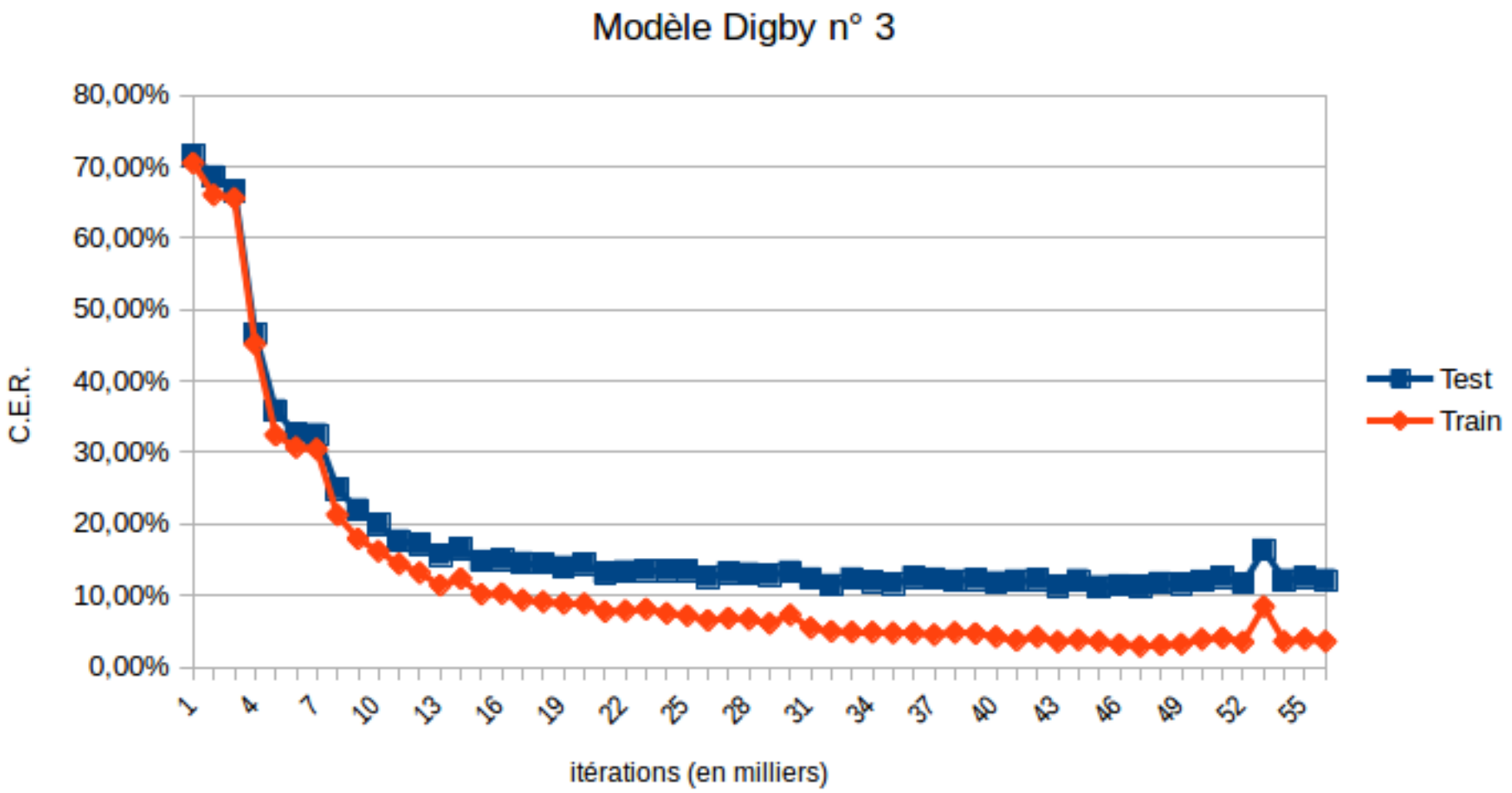
- approche non segmentées ;
- réseaux de neurones récurrents (LSTM) ;
- *open source* et nécessitant l'entraînement d'un modèle.

Un apprentissage guidé



Un apprentissage guidé

Entraînement sur un ms. (ici Roland d'Oxford)



Un apprentissage guidé

Résultats

- pour un imprimé ancien, des taux d'erreur de l'ordre de 1% sont atteignables...
- pour un ms., des taux inférieurs à 10% sont atteignables rapidement, >5% plus difficiles ;
- cas du Digby 23, env. 400 lignes d'entraînement, taux de succès de 89,16% en test, et 93% sur l'ensemble des données ;

Pour un modèle un peu plus général

https ://github.com/Jean-Baptiste-Camps/FROC-MSS

- 4 manuscrits différents du XII^{e}-XIII^{e} ;
- 3000 lignes env.
- CER de 8% en test (7% en ignorant l'espacement).

Évaluation et confusions fréquentes

```
Evaluating model_froc.mlmodel
Evaluating 100 %
=== report  ===

13645   Characters
1107    Errors
91.89 %  Accuracy

585     Insertions
156     Deletions
366     Substitutions

Errors  Correct-Generated
91      { SPACE } - {  }
67      { ı } - {  }
56      {  } - { SPACE }
36      { COMBINING ACUTE ACCENT } - {  }
32      { n } - {  }
```

T.P. Kraken : Préparation des données

1. Essayer d'appliquer un modèle déjà entraîné

```
# Lancer la reconnaissance
# et extraire un fichier de correction
$ ketos transcribe -o gt.html --prefill model_best.mlmodel
↪  tif/*.tif
# Ou, si ça vous satisfait, extraire directement un fichier texte
$ kraken -I "tif/*.tif" -o .txt segment ocr -m model_best.mlmodel
# ou hocr
$ kraken -I "tif/*.tif" -o .txt segment ocr -m model_best.mlmodel
```

Pas terrible... Mais comment améliorer le modèle ?

2. Corriger / transcrire

T.P. Kraken : préparer l'entraînement

3. Extraire et entraîner

```
#Extraire et normaliser les caractères
$ ketos extract --output book --normalization NFD gt.html
```

Ensuite idéalement, 80% des données vont servir à entraîner, 10% à évaluer les modèles pendant l'entraînement, 10% à tester le modèle retenu (on peut faire 90% - 10% si la précision de l'évaluation n'est pas critique).

- il est recommandé de faire cette répartition de manière aléatoire ;
- il est possible d'accroître artificiellement le nombre de lignes d'entraînement, en bruitant de différentes manières les lignes de GT.

Création des jeux `train`, `val` et `test`

Sélection des pages à transcrire

Lorsque vous préparez des données d'entraînement, il peut être préférable de ne pas transcrire des pages immédiatement contiguës, mais de les sélectionner aléatoirement dans l'ensemble du document.
→ éviter des biais dus à des changements de fonte, mise en page, etc.

Répartition en `train` / `val` / `test`

Pour des raisons similaires, et surtout pour des documents anciens ou irréguliers, il vaut mieux sélectionner aléatoirement des lignes de chaque page pour les séparer en `train` / `val` / `test`.

On peut créer un script Python ou autre pour ce faire :

```
$ python3 ../randomise_data.py book/*.png
```

Apparté : augmentation artificielle des données d'entraînement

Constat : transcrire coûte cher (temps, effort, ...).
Pour obtenir augmenter les données d'entraînement avec moins d'effort, on peut :

- générer des lignes arbitraires de textes, si l'on dispose de la fonte (commande `ketos linegen` dans Kraken) ;
- appliquer diverses déformations et bruiter les lignes (pseudo-tâches, trous, bavures, courbures, etc.)

Un bon outil : `Doccreator`, http ://doc-creator.labri.fr, qui permet de :

- extraire une pseudo-fonte à partir de documents ;
- multiplier les lignes par autant de déformations 2D ou 3D que l'on souhaite.

Permet en général des gains modestes (1 ou 2% de CER), mais appréciables.

T.P. Kraken : lancer l'entraînement

On peut ensuite lancer un entraînement (de zéro ou à partir du modèle précédent, aux choix).

```
#Lancer l'entraînement
$ ketos train -t train.txt -e val.txt
```

N.B : de nombreux autres paramètres sont disponibles, liés au modèle et à l'entraînement.
Si les données n'ont pas été normalisées auparavant, on peut utiliser l'option '-u NFD' (normalisation Unicode).

T.P. Kraken : Test des résultats

4. Tester les résultats

Une fois que l'entraînement a atteint un niveau satisfaisant, ou quand les paramètres de l'*early stopping* ont mené à l'interrompre (l'erreur a cessé de décroître suffisamment pendant un nombre donné d'époques) on peut tester la qualité du résultat,

```
# Test du meilleur modèle et confusions de caractères
$ ketos test -m model_best.mlmodel -e test.txt
```

Plan

Post-correction

Une fois que l'on a un modèle aux résultats satisfaisants, on peut l'appliquer à l'ensemble du document. Mais, quel que soit sa précision, il risque de rester tout de même des erreurs.
Il est délicat d'échapper à une relecture de l'ensemble, dans la plupart des cas, mais certains outils permettent d'accélérer ce processus :

- outils type 'correcteurs orthographiques' pour les langues modernes / standardisées ;
- modèles linguistiques un peu plus complexes pour les états anciens de langue.

Un exemple : `PoCoTo` (https ://github.com/cisocrgroup/PoCoTo).