Reconnaissance de caractères dans des textes manuscrits

Théo VIEL - Guillaume DESFORGES

Sommaire

1. Eléments de théorie
2. Reconnaissance avec **ocropy**
3. Création d’un réseau avec **Keras**
4. Comparaison des deux méthodes

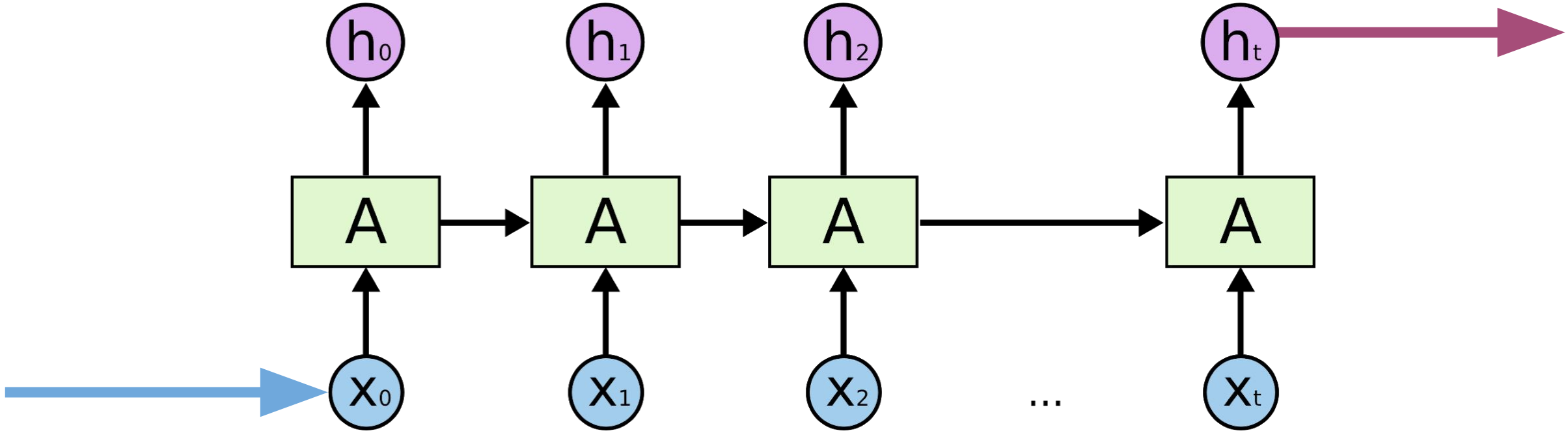
1. Eléments de théorie

Forme du problème

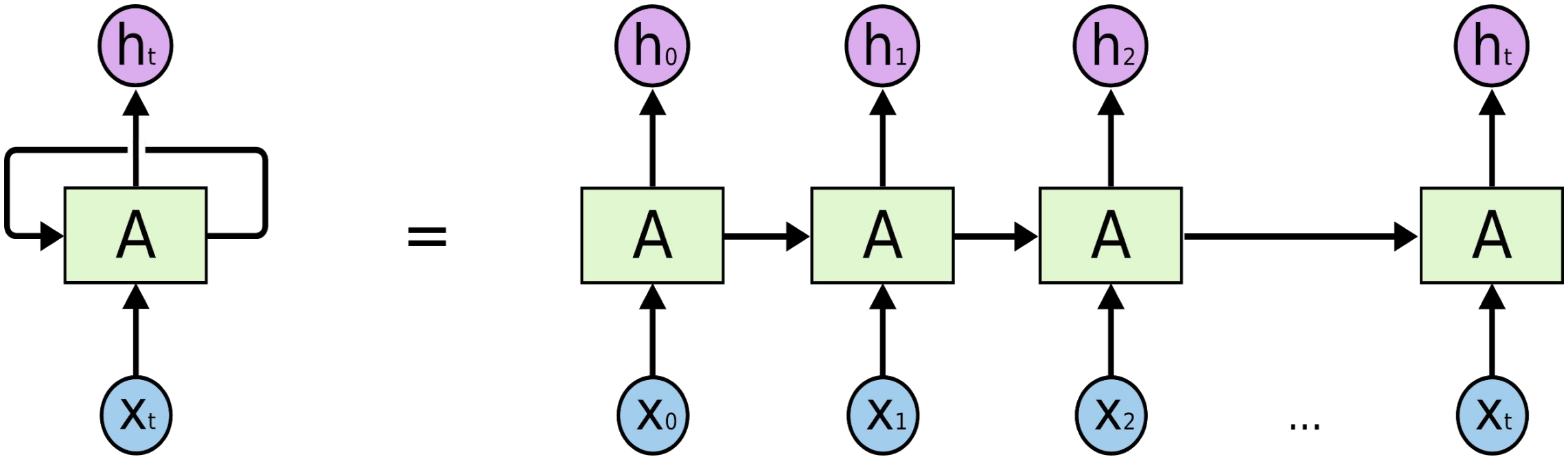
1. Input :
    - Images binarisée de longueur variable et de hauteur fixée,
    - Analogie temporelle avec une séquence audio ;
2. Technologies adaptées :
    - Les réseaux de neurones récurrents (RNN) tels que les LSTM ou GRU
    - La fonction de perte “*Connectionist Temporal Classification*” (CTC).

1.A. Réseaux de neurone récurrent (RNN)

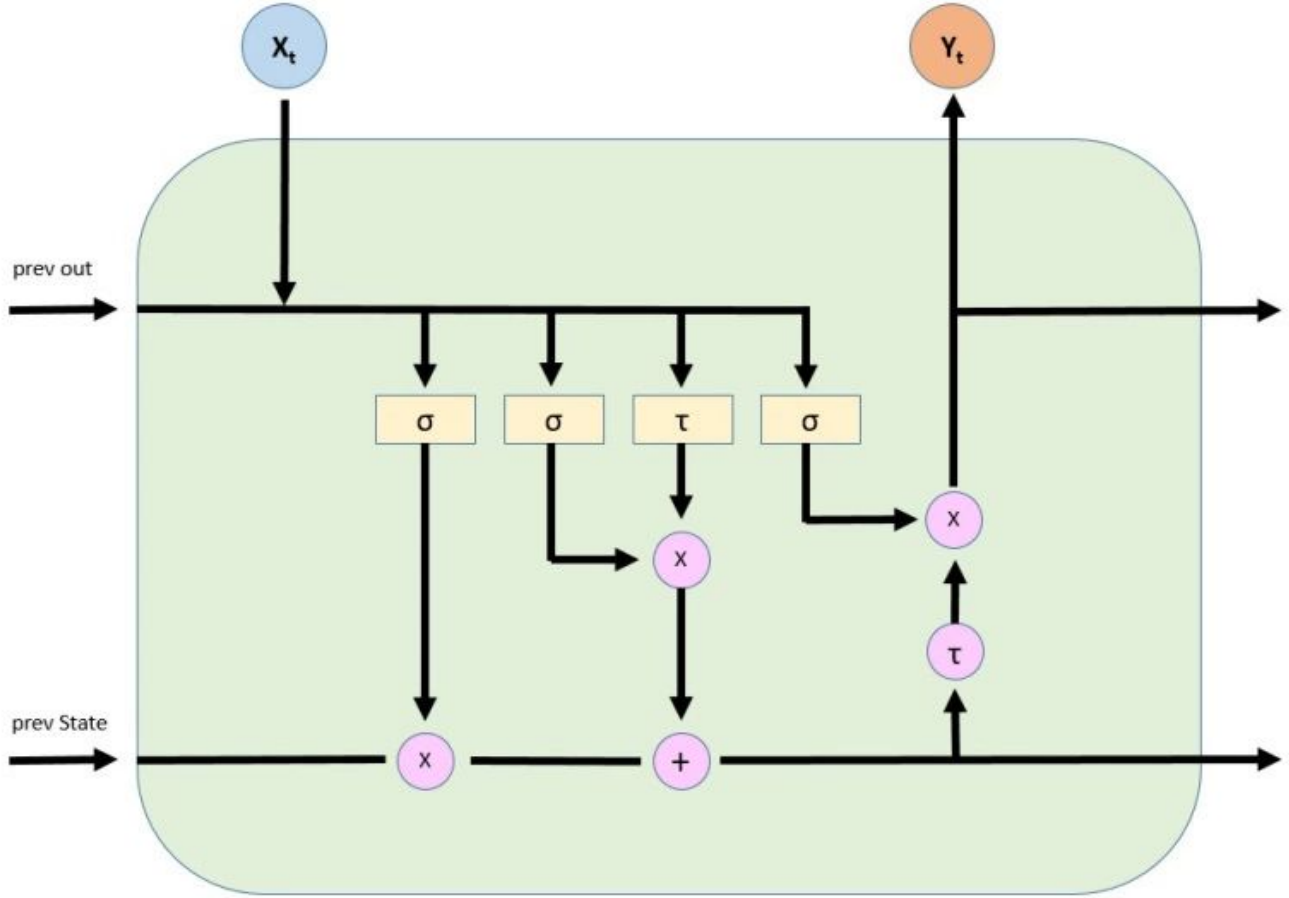
1. Traiter des données séquentielles
    - comprendre des modèles séquentiels
    - temporelles ou spatiale
    - multiplicité des outputs
2. Fonctionnement :
    - une “cellule” calcule un output et un état à partir d’un input et d’un état précédent
    - processus répété sur toute a longueur de la séquence
    - on peut récupérer l’output à chaque pas de la séquence, ou l’output final uniquement



Traiter une séquence avec un NN : trop profond !



Solution : une seule unité, une seule couche



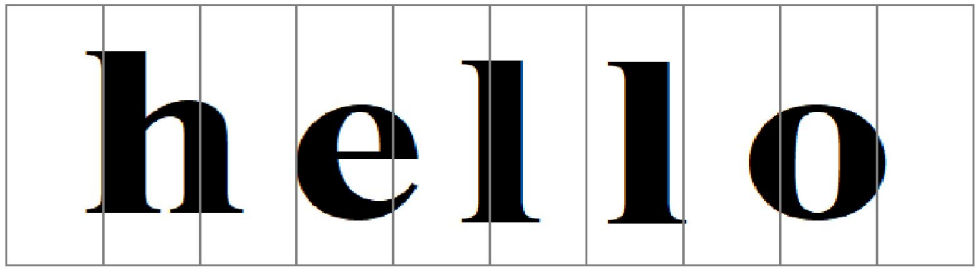
Exemple de cellule RNN : le LSTM

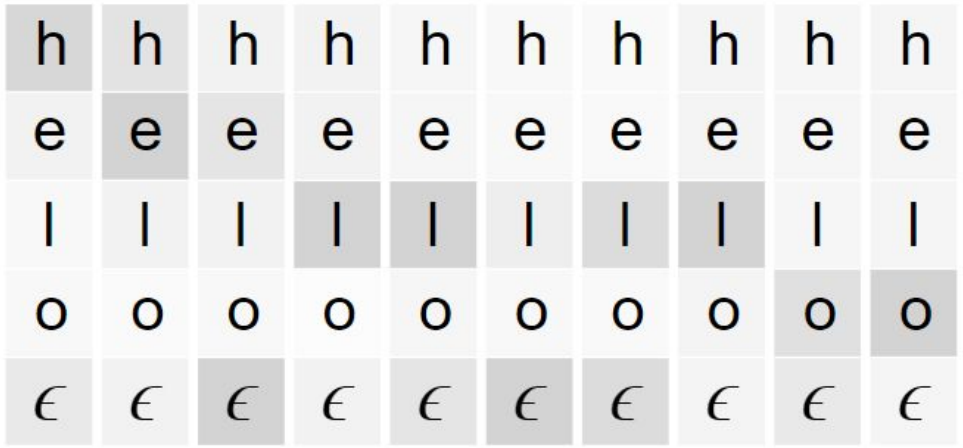
1.B. *Connectionist Temporal Classification Loss*

Objectif : comparer la séquence prédite à la vraie séquence

- On veut *a priori* classifier chaque colonne de pixels
- Cependant, une colonne n'équivaut pas à une lettre
- Loss adaptée : la **CTC**
- différentiable et rapide à calculer ?

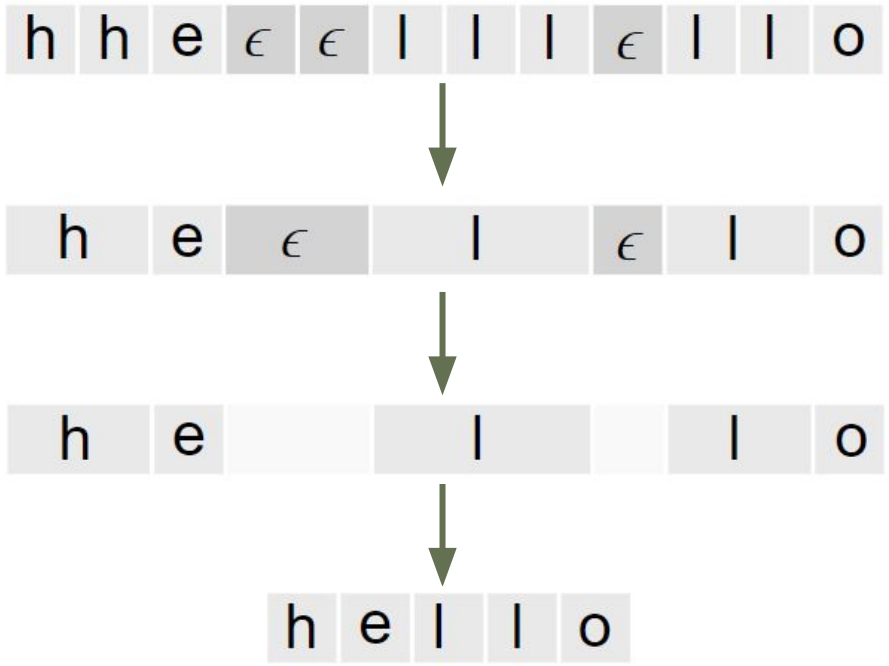
Obtention des alignements





- Séparation de l'image en colonnes ;
- Calcul des probabilités pour chaque colonne ;
- Obtention des séquence de caractères :
  - "Alignements"

Traduction naïve des alignements



1. On choisit l'alignement le plus probable ;
2. On applique l'algorithme de *merging* ci-contre :
3. On obtient un label.

Problème : ce label n'est pas forcément celui le plus probable.

Définition de la fonction de perte CTC

- On s'intéresse à la probabilité d'obtenir le bon label `l` pour l'image d'entrée `x` :

$$P(l|x) = \sum_{\pi \in \mathcal{B}(l)} P(\pi|x)$$

  - `B(l)` est l'ensemble des alignements donnant le label `l` :
  - Sous l'hypothèse d'indépendance, on a :

$$P(\pi|x) = \prod_{t=0}^{T} P(y_t = \pi_t|x)$$

- On applique ensuite le principe du **maximum du log de vraisemblance** sur cette probabilité:
  - Ceci implique de calculer les probabilités de tous les alignements donnant le label `l` .

Calcul des probabilités des labels

- On calcule les probabilités par **programmation dynamique** en utilisant les préfixes:
  - On définit la probabilité d'avoir le préfixe de longueur `s` à l'instant `t`:

$$\alpha_t(s) = P(l_{1:s}|x_{1:t})$$

- On commence par définir un label `l'` en ajoutant des blank labels à `l` :

h e l l o -> ε **h** ε **e** ε **l** ε **l** ε **o** ε

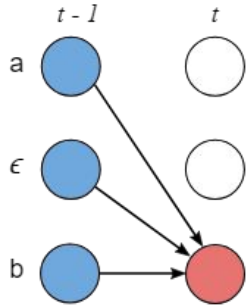
- On initialise de la façon suivante:

$$\alpha_1(1) = P(y_1 = \epsilon), \quad \alpha_1(2) = P(y_1 = l_1), \quad \alpha_1(s \geq 2) = 0$$

Calcul des probabilités des labels

L'équation de récurrence s'écrit suivant 2 cas :

1. 3 façons de traduire l'élément `b` :
   - `. b`
   - `ε b`
   - `b b`



$$\alpha_t(s) = P(y_t = l'_t|x)\Big(\alpha_{t-1}(s-2) + \alpha_{t-1}(s-1) + \alpha_{t-1}(s)\Big)$$

2. Cas particuliers:
   a. Traduire `ε` :
      - `. . ε`
      - `. ε ε`
   b. Traduire `b b`
      - `. b ε b`
      - `b ε b b`

$$\alpha_t(s) = P(y_t = l'_t|x)\Big(\alpha_{t-1}(s-1) + \alpha_{t-1}(s)\Big)$$

Vers le maximum de vraisemblance

Notre fonction objectif est : $\mathcal{L} = -\sum_x ln\big(p(l|x)\big)$

On a besoin de calculer $\frac{\partial \mathcal{L}}{\partial P(y_t = k)}$

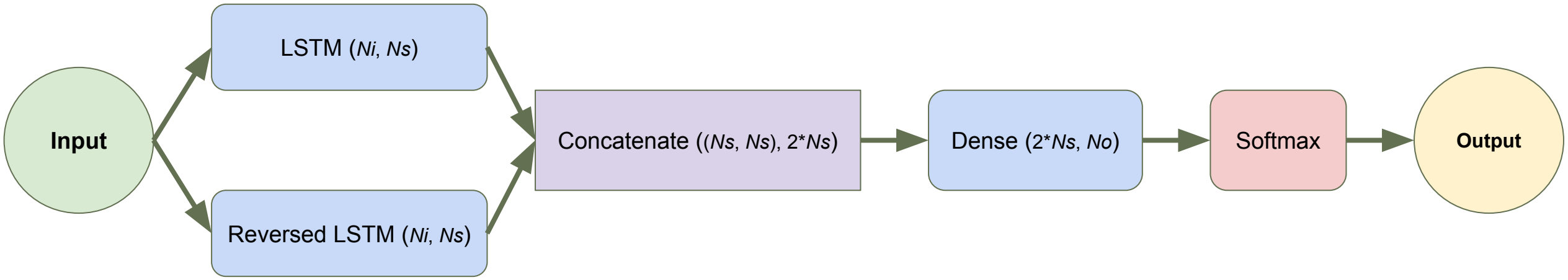
- On utilise pour cela $\alpha_t(s)$ défini précédemment et $\beta_t(s)$ défini sur les suffixes.

Cette quantité peut s'exprimer en fonction de :

- $\alpha_t(s)$
- $\beta_t(s)$
- $P(y_t = k|x)$

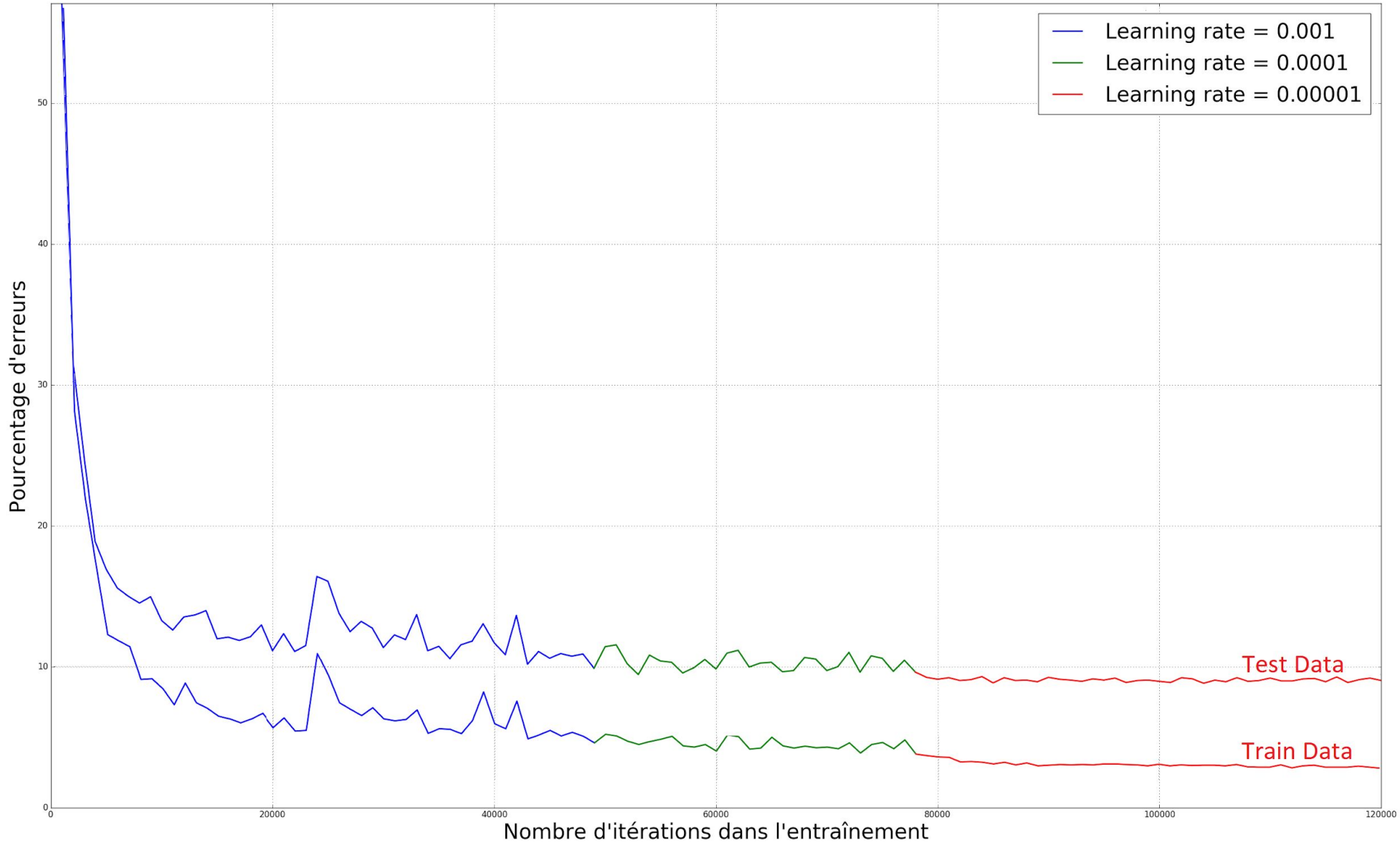
2. Reconnaissance avec ocropy

Le réseau utilisé

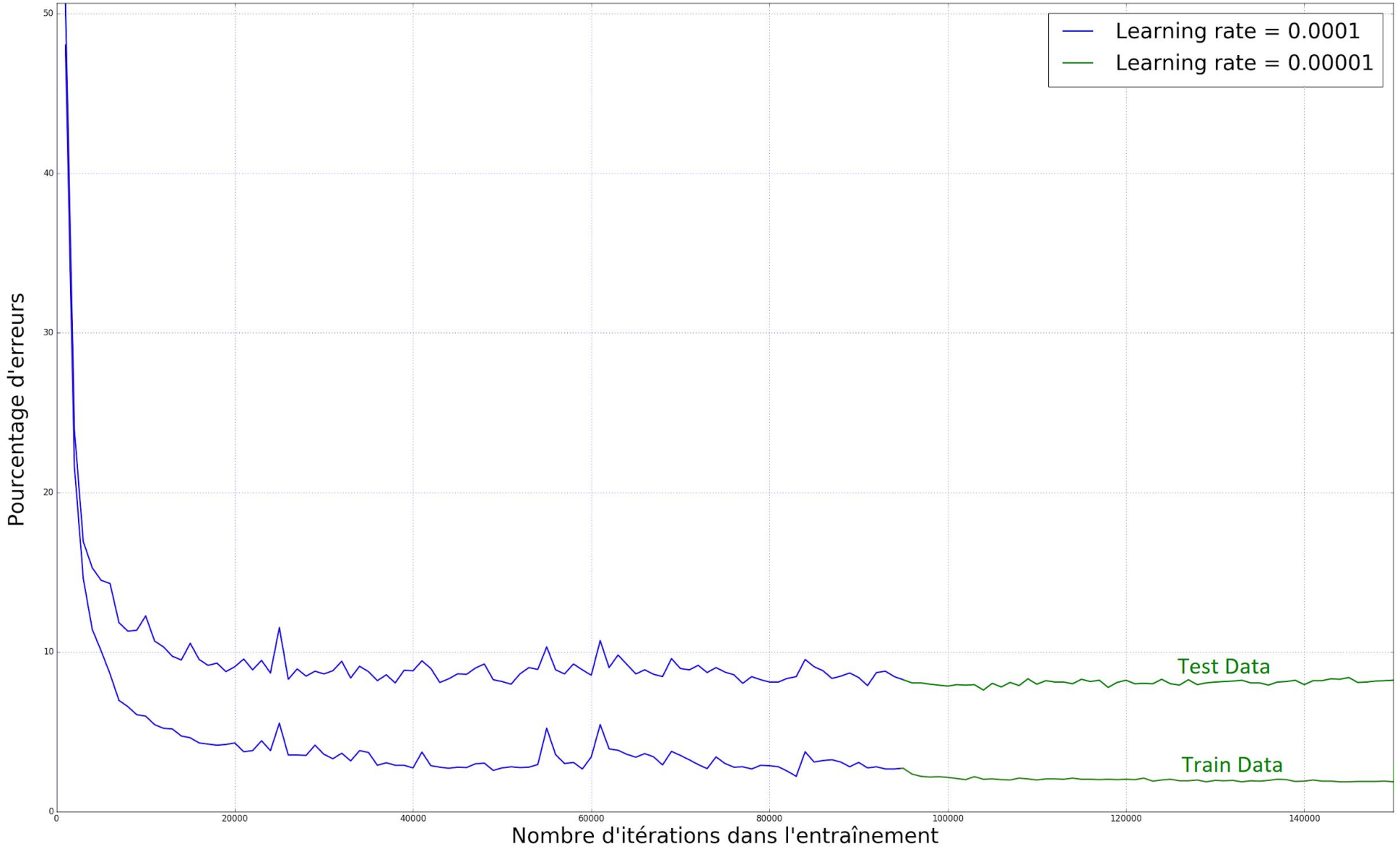


Utilisation

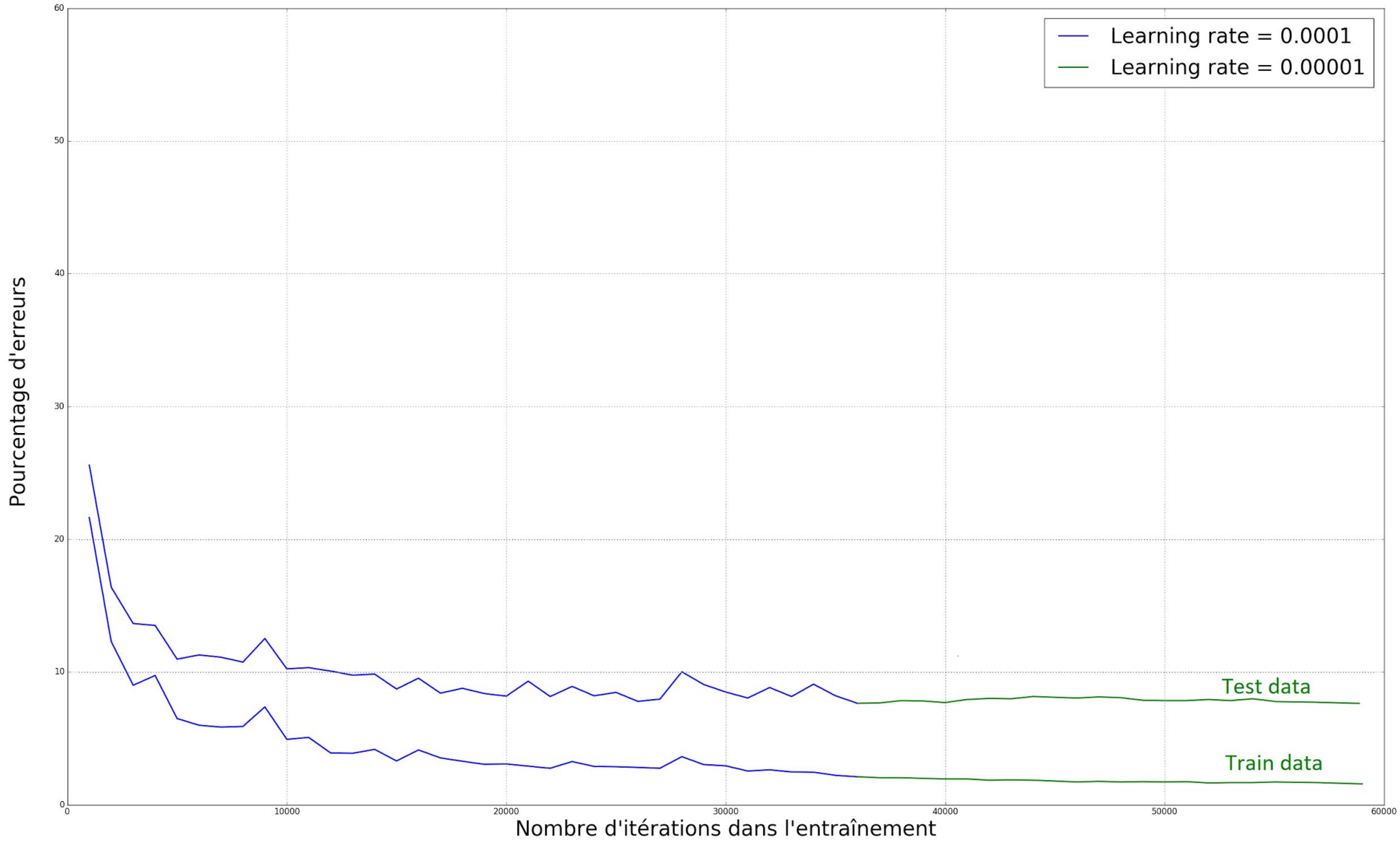
1. Entraînement avec les caractères spéciaux :
   - Ajout manuel dans le fichier chars.py
2. Commande utilisée :
   - *ocropus-rpred -m <ModelName> test/*/*.bin.png*
3. Possibilité de :
   - Charger un modèle
   - Changer la *learning rate*
   - Changer la taille de cellule



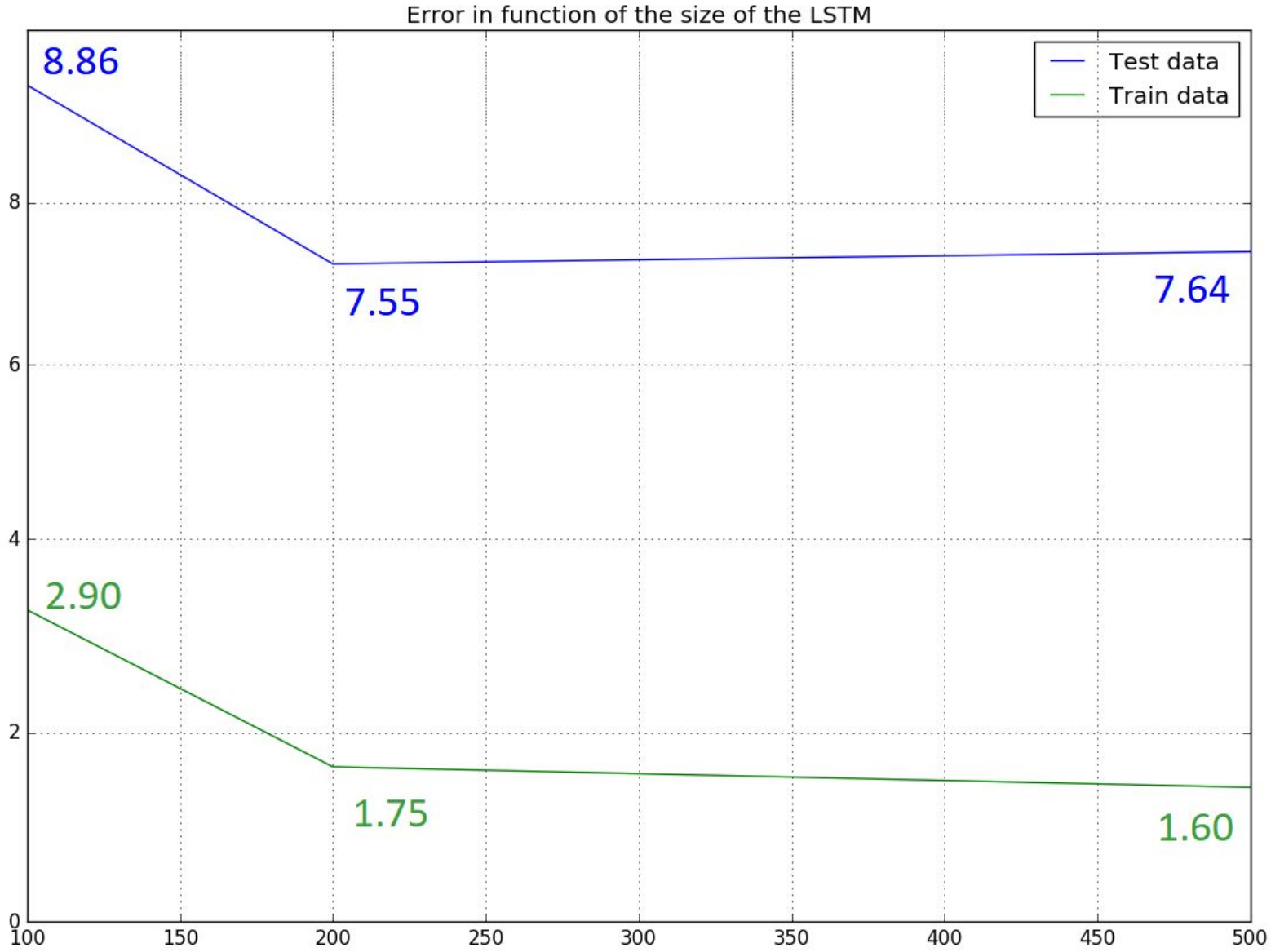
Courbes d’erreurs pour une cellule de taille 100



Courbes d’erreurs pour une cellule de taille 200



Courbes d’erreurs pour une cellule de taille 500



Erreur minimale obtenue pour chaque taille de cellule

Parenthèse sur **Tesseract**

1. OCR développé par HP, repris par Google, Gratuit , Open Source;
2. Très efficace pour des modèles déjà existants:
   - Entraîné par Google sur de gigantesques datasets;
3. Mais pas de modèle existant pour les langues des manuscrits:
   - Il faut entraîner notre propre modèle :
     . On repart à zéro
     . Documentation lourde, peu digeste;
   - Nous disposons de peu de données/moyens par rapport à Google:
     . Temps d'entraînement trop long,
     . Risque de surentraînement.

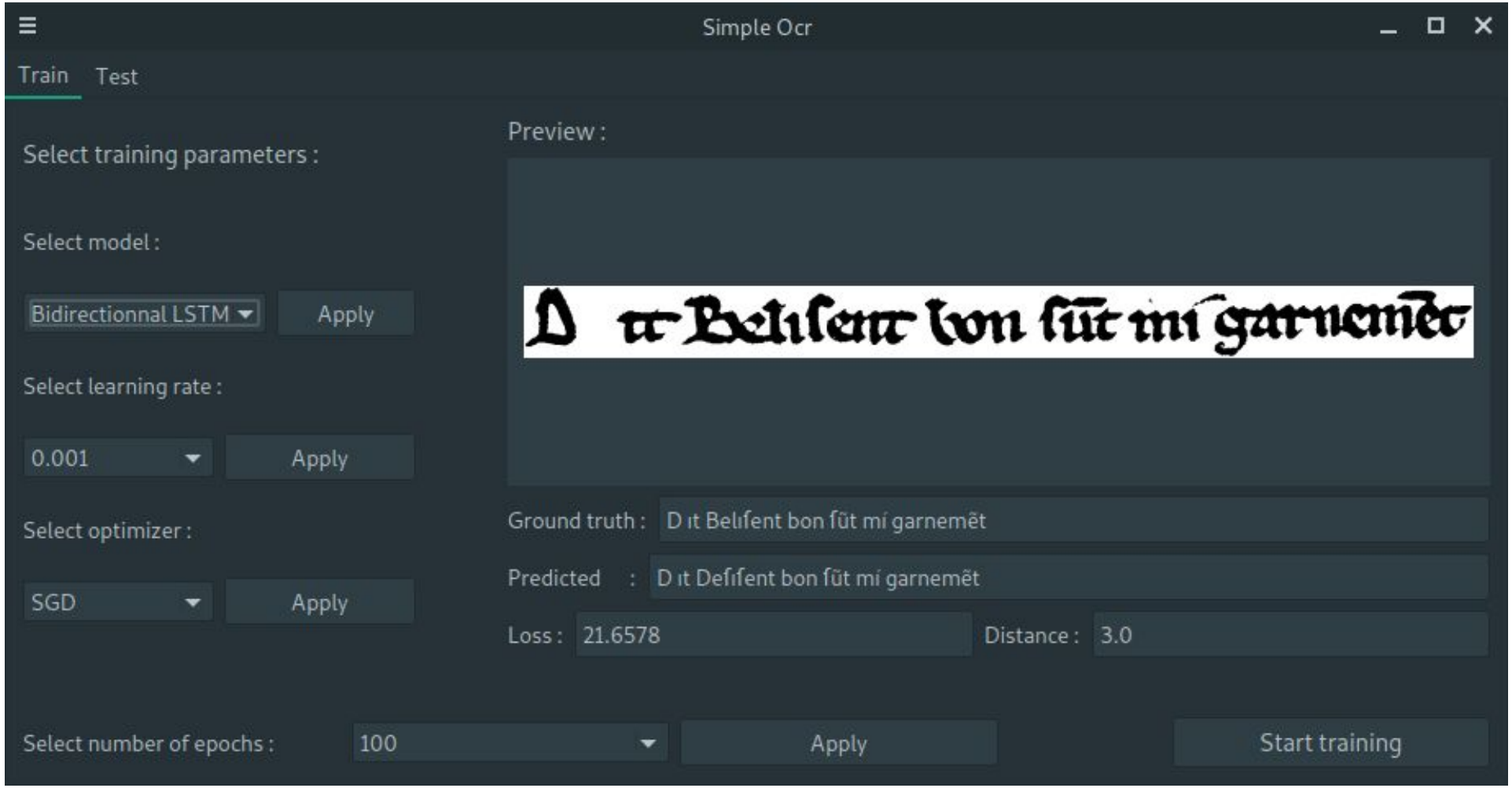
3. Création d’un OCR

Objectifs

Implémenter un OCR **fonctionnel** et **simple d'utilisation**

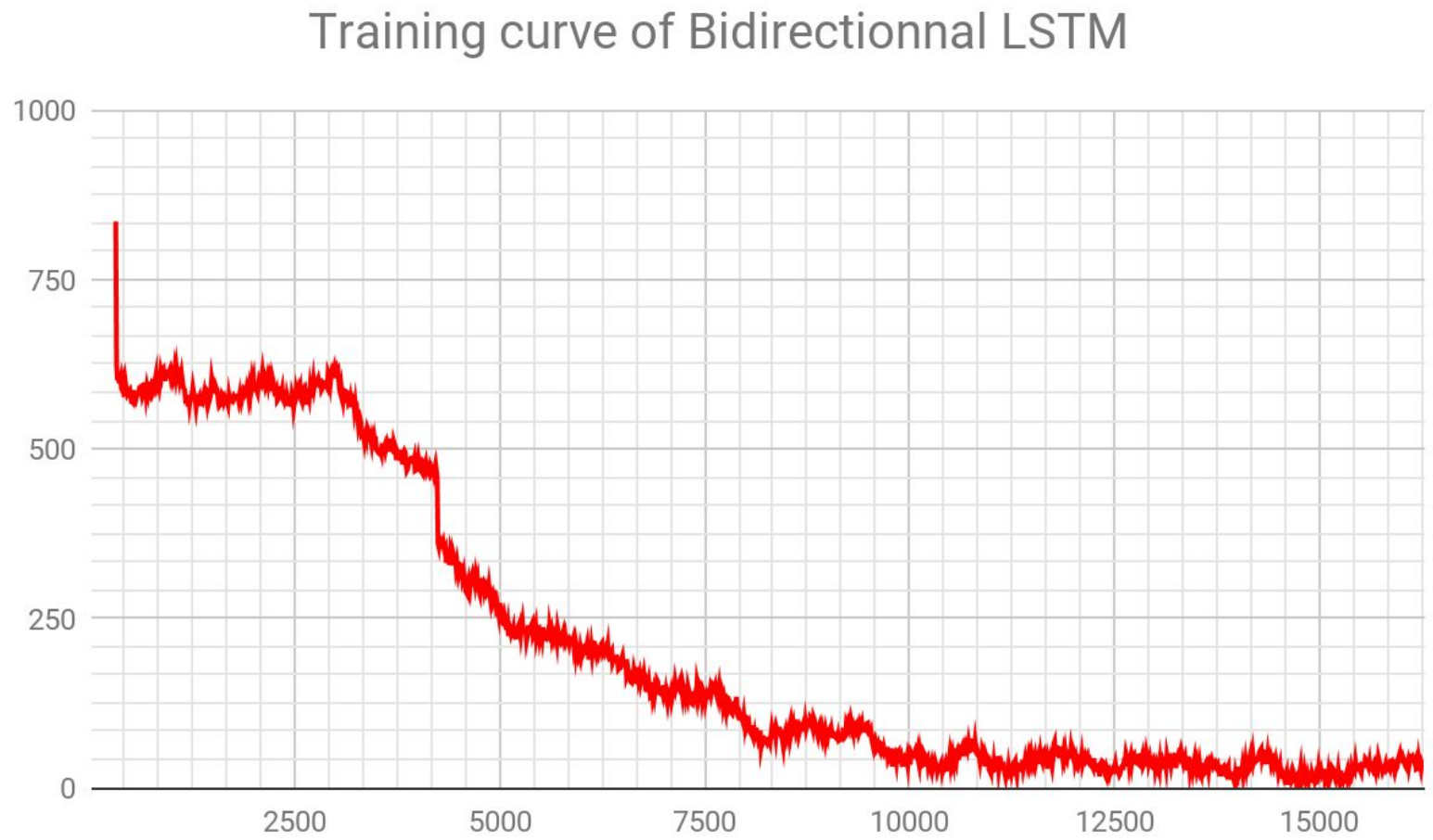
→ *Entraîner* un modèle

→ Utiliser un modèle pour *prédire*



Interface graphique - exemple d'entraînement

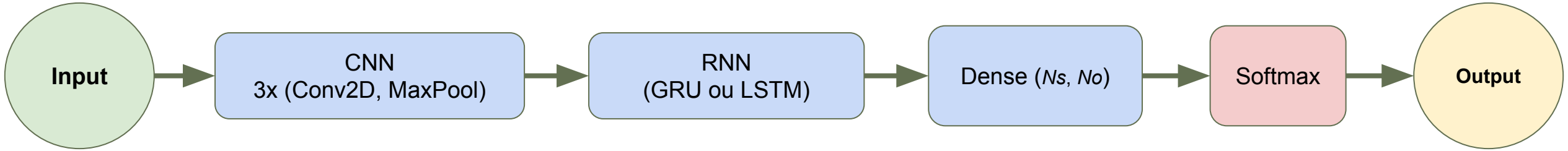
Interface graphique - exemple de prédiction



Exemple d’apprentissage

Difficultés majeures

- Compréhension d'une librairie de Deep Learning
  - Usage de CTC loss/decode peu (pas) documenté
- Compréhension du code source d'ocropy
- Développement logiciel
- Threader l'apprentissage (GUI vs TensorFlow)



Une autre architecture : CNN+RNN

Expériences

L’architecture de réseau ***Bidirectional LSTM***

- plus dure à entraîner
- hyper paramètres à choisir plus finement

L’architecture de réseau ***CNN+RNN***

- plus rapidement de bons résultats
- overfit plus souvent
- *rapproche les variations informatives ?*

Conclusion