



Digital Humanities Workshops - HS 2022
Bern Switzerland

Advance Text Recognition using PyLaia Toolkit Practice Session

By

Alejandro H. Toselli and Joan Andreu Sánchez

[ahector, jandreu]@prhlt.upv.es

Pattern Recognition and Human Language Technology Research Center
Universitat Politècnica de València



tranSkriptorium



UNIVERSITAT
POLITÈCNICA
DE VALÈNCIA

November 29th, 2022

1 Introduction: PyLaia Toolkit

PyLaia¹ is a device agnostic, PyTorch² based, deep learning toolkit specialized for handwritten text recognition (HTR). It is the successor of **Laia** ToolKit.³ Mostly developed by Joan Puigcerver and Carlos Mocholí (both from the Universitat Politècnica de València), **PyLaia** implements a *Deep Convolutional Recurrent Neural Network* (CRNN) trained with *Connectionist Temporal Classification* (CTC). The main improvements of **PyLaia** with respect to **Laia** are:

- *more efficient*: in terms of RAM resource utilization and processing time.
- *portability*: source code written in **Python** and based on the deep-learning **PyTorch** platform.
- running on both CPU and GPU hardware (with multi-GPU capability).

PyLaia is available at the **Transkribus** platform.⁴ As displayed in Fig. 1, in the **Transkribus** user interface **PyLaia** is placed into the Tab “Tools”, as an option of the “Test Recognition” selection-list (the CITlab’s HTR tool is the other option). Actually, **PyLaia** has similar functionalities and performance characteristics as the CITlab’s HTR approach, but the former is completely open source.

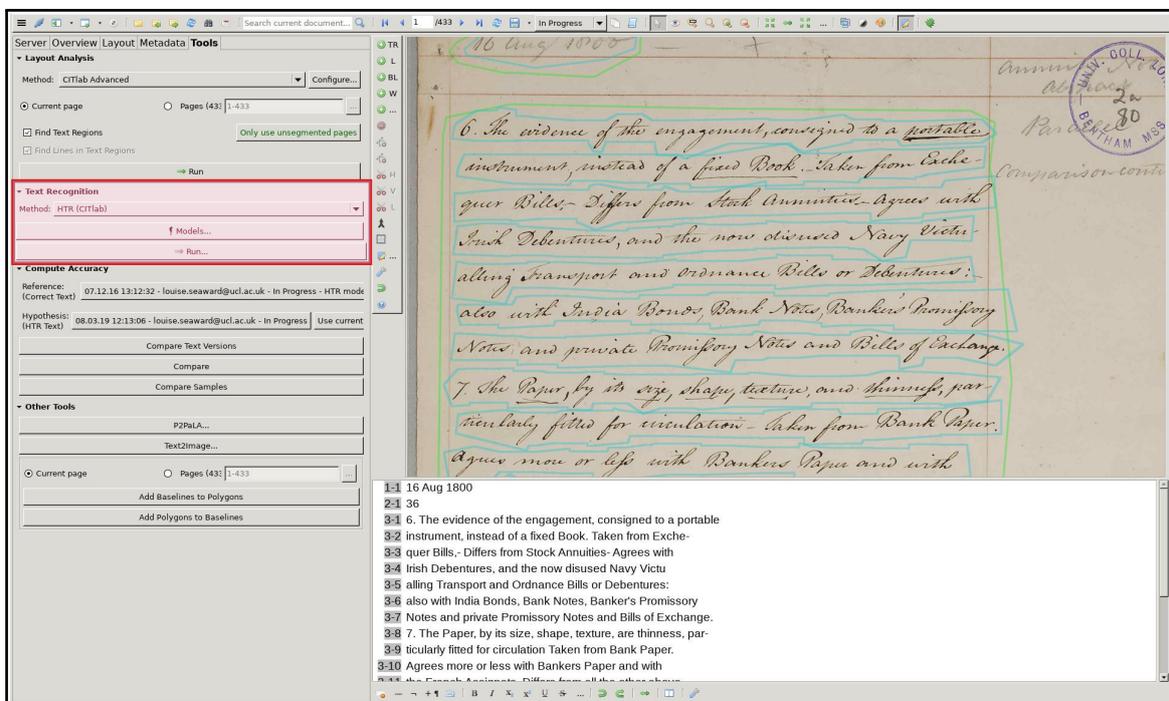


Figure 1: Screenshot of the **Transkribus** interface. The area highlighted in red shows the “text recognition” options in the Tab “tools”.

It is worth mentioning that the work [1] reports a benchmark of different open-source *Handwritten Text Recognition* systems (including **PyLaia**), which were tested on a dataset of handwritten historical documents in Norwegian. The recognition accuracy of **PyLaia**’s predictions are in the top three reported.

¹<https://github.com/jpuigcerver/PyLaia>

²<https://pytorch.org>

³<https://github.com/jpuigcerver/Laia.git>

⁴<https://transkribus.eu/Transkribus/>

2 Installing PyLaia

The source code of **PyLaia** is freely available at: <https://github.com/jpuigcerver/PyLaia>, the popular **GitHub**⁵ hosting platform for software development version control using `git`.⁶

There are several ways to install this Toolkit. In the following section, three ways are briefly described. For the first two, it is assumed that we are working on a Linux system (**Ubuntu**⁷ or any other Debian-based distribution) running on a machine equipped with a *Graphic Processing Unit* (GPU).

2.1 Installing PyLaia on Local System using Python Standard Library

For this case, it is required that **python3** and **pip** packages to be already installed in the local system. These are the steps to install **PyLaia** in the local system:

```
# Install "python", "pip" and "git" in case they are not in the system
apt-get install -y python3 pip git

# This version fixes an issue with the required package jsonargparse
pip install jsonargparse==4.13.0

# Download PyLaia from the GitHub repository
git clone https://github.com/jpuigcerver/PyLaia

# Install the PyLaia toolkit
cd PyLaia
pip install -r requirements.txt
python setup.py install
```

2.2 Installing PyLaia on Local System using Python Virtual Environment

This case requires that **virtualenv** package was previously installed. The installing steps are as follows:

```
# Install "virtualenv" in case it has not already done
apt-get install -y virtualenv

# Download PyLaia from the GitHub repository
git clone https://github.com/jpuigcerver/PyLaia

# Create a python virtual enviroment named "RDNN-HTR-PY"
virtualenv -p python3 RDNN-HTR-PY

# Activate the virtual environment
source RDNN-HTR-PY/bin/activate

# This version fixes an issue with the required package jsonargparse
pip install jsonargparse==4.13.0
```

⁵<https://github.com>

⁶<https://git-scm.com>

⁷<https://ubuntu.com>

```
# Install the PyLaia toolkit
cd PyLaia
python setup.py install
```

The advantage of this installing method is that everything is placed inside the directory “RDNN-HTR-PY” and avoids conflicts with our host Linux distribution. To disable the virtual environment, just run “deactivate” at the shell prompt.

2.3 Installing PyLaia on Anaconda Platform (recommendable option)

Anaconda platform is highly recommended for installing all the required software. Check out the official website at <https://anaconda.org> for instructions about how to install the [Miniconda](#) package. Basically, to install it:

```
wget \
https://repo.anaconda.com/miniconda/Miniconda3-latest-Linux-x86_64.sh
bash Miniconda3-latest-Linux-x86_64.sh
```

Thereupon, the PyLaia ToolKit can be installed as follows:

```
# Create a directory for placing stuff
mkdir CONDA-SRC

# Clone the PyLaia repository on the local machine
git clone https://github.com/jpuigcerver/PyLaia CONDA-SRC/PyLaia

# Create a new conda environment for PyLaia
conda create -n PyLaia python=3.7.11
conda activate PyLaia

# This version fixes an issue with the required package jsonargparse
pip install jsonargparse==4.13.0

# Installing PyLaia and remaining dependencies
pip install -e CONDA-SRC/PyLaia

# To remove PyLaia environment from CONDA
# conda deactivate
# conda env remove -n PyLaia
```

2.4 Installing and Using PyLaia on Google Colab Platform

To use **PyLaia** on *Google Colaboratory Platform* (*Colab* for short), a compliant web browser (Firefox, Chrome, MS Edge, etc.) is required to connect to “Google Drive” at: <https://drive.google.com/drive/my-drive> There, it will be asked us to enter our Google *login* and *password* account. If the “add-on” *Colab* is not already enable, we have to click on the button tagged with **[+New]** (top left), and then click on **[More]** and select **[Google Colaboratory]** to install it.

To start the practice session in *Colab*, we first upload the provided file `Prac_PyLaia.ipynb` (https://www.prhlt.upv.es/~ahector/BERN/NoteBook/Prac_PyLaia.ipynb) to *Goggle Drive* by clicking again on the button **[New]** and then on the **[File upload]**. Next, a pop-up window will appear prompting to select a file to upload.

Once the file `Prac_PyLaia.ipynb` has been uploaded, it will be displayed in *My Drive*. By clicking on this filename, *Colab* is started and then the file `Prac_PyLaia.ipynb` is loaded. So, now we have everything ready to start this practice in *Colab*.

3 Installing other required Tools

As will be seen later on, for extracting and processing line images from page images using the information contained in corresponding PAGE XML files [2], we employ the open-source tool **textfeats**, which can be downloaded at: <https://github.com/mauvilsa/textfeats>. For instructions about its requirements and how to install it, refer package documentation.

To display **PyLaia**'s predictions of line images along with their word segmentation, we will use the python script `visualize_segmentation.py`, which can be downloaded at: https://www.prHLT.upv.es/~ahector/BERN/Aux/visualize_segmentation.py.

Another (generic) tool that will be required for parsing XML files is **xmlstarlet**. This will be used for processing the PAGE XML files included in the dataset described in the next section.

4 Experimental Dataset and Performance Measures

The handwritten text dataset employed for trying **PyLaia** and the performance measures for evaluating recognition results are presented below.

4.1 ICFHR'14 Bentham Dataset

This dataset was already used in the ICFHR 2014 Handwritten Text Recognition contest.⁸ This is a subset of manuscripts taken from the Bentham Papers written by the English philosopher and reformer Jeremy Bentham⁹ and some copies written by his secretarial staff. The manuscripts covers different subjects as legal reform, punishment, the constitution, religion, and his panopticon prison scheme. Fig. 2 shows some examples of the manuscript images included in this dataset.

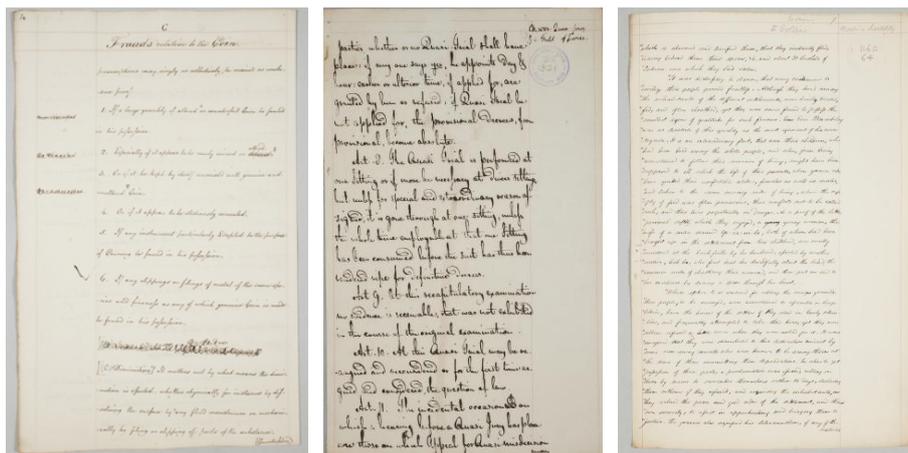


Figure 2: Document samples of the ICFHR-2014 dataset.

The dataset is available for research purpose at ZENODO: <https://zenodo.org/record/44519>. The GT is provided in PAGE XML format [2]. Table 1 summarizes the basic statistics of this dataset. Page

⁸<http://transcriptorium.eu/~htrcontest>

⁹<http://blogs.ucl.ac.uk/transcribe-bentham/jeremy-bentham>

images and transcripts of training and validation partition sets are provided both at page level and at line level, while page images of the test partition set are provided only at line level.

Table 1: Main statistics of the ICFHR-2014 dataset. The images were scanned at 300dpi.

Number of:	Training	Validation	Test	Total
Pages	350	50	33	433
Lines	9 198	1 415	860	11 473
Running words	86 075	12 962	7 868	106 905
Lexicon	8 658	2 709	1 946	9 716
Character set size	86	86	86	86
Running characters	442 336	67 400	40 938	550 674

For more details about this dataset (data organization, format of transcript and image files, HTR baseline results, etc.), refer to [3, 4] and the documents within it.

4.2 Evaluation Measures for HTR

Once the recognition has finished, we can assess the accuracy of the recognized hypotheses through the *Character Error Rate* (CER) and the *Word Error Rate* (WER) measures. WER/CER is defined as the minimum number of words/characters that need to be substituted, deleted, or inserted to match the recognition outputs with the corresponding reference ground truths, divided by the total number of words/characters in the reference transcripts:

$$\text{WER/CER} = \left(\frac{D + S + I}{N} \right) \cdot 100$$

where:

I : number of word/character insertions

D : number of word/character deletions

S : number of word/character substitutions

H : number of word/character correct labels

N : total number of word/character references in the defining transcription files.

Fig. 3 illustrates obtained CER/WER figures for two samples from the ICFHR-2014 benchmark.

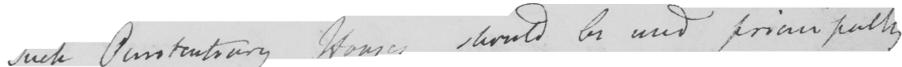
<p>I </p> <p>H such Penstentrery Hoases should be anid priapalty</p> <p>R such Penitentiary Houses should be and principally</p>	<p>WER = 4/7 = 57%</p> <p>CER = 8/50 = 16%</p>
<p>I </p> <p>H for eomfromiy and employing in hard lebour , Persons</p> <p>R for confining and employing in hard labour , Persons</p>	<p>WER = 2/9 = 22%</p> <p>CER = 8/52 = 15%</p>

Figure 3: Two examples of test line images (I), automatic (H) and reference (R) transcripts, along with the corresponding WER and CER.

For computing WER/CER we will utilize the python package [editdistance](#).

5 Data Preparation

5.1 Line Extraction and Processing

This section describes the way to extract and process text line images (along with their transcripts) from page images. Although, extracted & processed lines and their transcripts are already provided at the ZENODO website.

From the already created `WorkDir` directory, containing the `BenthamDatasetR0-GT.tbz` and `BenthamDatasetR0-Images.tbz` files downloaded from ZENODO at <https://zenodo.org/record/44519>, the following shell-script commands are executed:

```
mkdir -p $HOME/WorkDir/DATA && cd $HOME/WorkDir/DATA
# Decompress both "tbz" files into the DATA directory
tar xvjf ../BenthamDatasetR0-GT.tbz
tar xvjf ../BenthamDatasetR0-Images.tbz
# Create a symbolic link of BenthamDatasetR0-GT/Partitions
ln -s BenthamDatasetR0-GT/Partitions
# Put together page images and PAGE XML files within IMAGE dir
mkdir IMAGES; cd IMAGES
cp -s ../BenthamDatasetR0-Images/Images/Pages/*.jpg .
cp -s ../BenthamDatasetR0-GT/PAGE/*.xml .
cd ..
```

As commented before, the `textfeats` tool [5] is employed to extract line from page images using their location coordinates included in corresponding PAGE XML files, and apply then some handwriting style attribute correction/normalization on them. The setup configuration for this tool is shown below (it is assigned to the shell variable `textFeats_cfg`):

```
textFeats_cfg='
TextFeatExtractor: {
  verbose      = false;           // Whether to be verbose
  deslope     = true;            // Whether to do automatic desloping of the text
  deslant     = true;           // Whether to do automatic deslanting of the text
  type        = "raw";          // Type of feature to extract, either "dotm" or "raw"
  format      = "img";          // Output features format, either "htk", "ascii" or "img"
  stretch    = true;           // Whether to do contrast stretching
  enh         = true;           //
  enh_win     = 30;             // Window size in pixels for local enhancement
  enh_prm     = 0.2;            // Sauvola enhancement parameter
  //enh_prm   = [ 0.05, 0.2, 0.5 ]; // 3 independent enhancements, each in a color channel
  normheight  = 0;              // Normalize image heights
  normxheight = 0;              //
  momentnorm  = true;           // Global line vertical moment normalization
  fpgram      = false;          // Whether to compute the features parallelograms
  fcontour    = true;           // Whether to compute the features surrounding polygon
  fcontour_dilate = 0;          //
  padding     = 10;             // Padding in pixels to add to the left and right
};
```

The script below extracts and processes line images for each XML PAGE file (and corresponding page image in JPG format) using the configuration stored in the shell variable `textFeats_cfg`. The processed lines are then placed into the (previously created) `Lines` directory:

```
mkdir Lines
for i in IMAGES/*.xml; do
  N=`basename $i .xml`
  textFeats --cfg <( echo "$textFeats_cfg" ) --outdir Lines $i
done
```

We assume that the `textFeats` binary location has been added to the system path.

The following shell script extracts line transcripts along with their line IDs and puts them into the `Transcripts.txt` file.

```
for f in IMAGES/*.xml; do
  xmlstarlet sel -t -m '//_:TextLine' -v ../../@imageFilename \
    -o '.' -v @id -o " " -v _:TextEquiv/_:Unicode -n $f
done | sed -r "s/\.jpg//" > Transcripts.txt
```

The format of produced `Transcripts.txt` file is as follows:

```
# Format
# Line-ID          <word1> <word2> <word3> ...
...
035_322_001.r186  S. 2
035_323_001.r5   182
035_323_001.r205 3
035_323_001.r6   Constitutional Code
071_010_002.r47  (B) (C)
...
```

To generate the lists of required line IDs: `TrainLines.lst`, `ValidationLines.lst` and `TestLines.lst` from the corresponding lists of page IDs: `Train.lst`, `Validation.lst` and `Test.lst`:

```
grep -f Partitions/Train.lst Transcripts.txt |
cut -d " " -f1 | sort -u > TrainLines.lst

grep -f Partitions/Validation.lst Transcripts.txt |
cut -d " " -f1 | sort -u > ValidationLines.lst

grep -f Partitions/Test.lst Transcripts.txt |
cut -d " " -f1 | sort -u > TestLines.lst

mv TrainLines.lst ValidationLines.lst TestLines.lst Partitions/
```

5.2 Preparing Data for using with PyLaia

We assume here that the directory `WorkDir` exists and contains the sub-directories `Lines` (with the processed line image files) and `Partitions` (with the files of lists with training & validation line IDs) and the file `Transcripts.txt`. First, we proceed to create some symbolic links inside the working directory `WorkDir`:

```
cd $HOME/WorkDir
ln -s ./DATA/Transcripts.txt
ln -s ./DATA/Lines
ln -s ./DATA/Partitions
```

From the files with the lists of training and validation line IDs: `Partitions/TrainLines.lst` and `Partitions/ValidationLines.lst`, and the file containing the (word level) transcripts `Transcripts.txt`, we generate corresponding character level transcription files `train_gt.txt` and `valid_gt.txt`. To carry out this task, we have the following the shell script:

```

# Creating train_gt.txt file
gawk 'BEGIN{
    while (getline < "Partitions/TrainLines.lst" > 0) T[$1]="
    ){ if (($1 in T)) { print $0} }' Transcripts.txt |
gawk '{ key=$1; $1=""; L=length($0); printf key;
    for (l=1;l<=L;l++) {
        c=substr($0,l,1);
        if (c==" ") printf " <space>"; else printf " "c;
    } if (L==0) print " <space> <space>"; else print " <space>"
}' > train_gt.txt

# Creating valid_gt.txt file
gawk 'BEGIN{
    while (getline<"Partitions/ValidationLines.lst">0) T[$1]="
    ){ if (($1 in T)) { print $0} }' Transcripts.txt |
gawk '{ key=$1; $1=""; L=length($0); printf key;
    for (l=1;l<=L;l++) {
        c=substr($0,l,1);
        if (c==" ") printf " <space>"; else printf " "c;
    } if (L==0) print " <space> <space>"; else print " <space>"
}' > valid_gt.txt

```

An this is the Python script version of the previous shell script (also used in the *Colab* platform):

```

# Creating train_gt.txt file
with open("Transcripts.txt") as f, \
    open("Partitions/TrainLines.lst") as g:
    trLst=set((x.strip()) for x in g)
    trLbs=dict( \
        ( x.split(' ',1)[0].strip(), \
          re.sub(r'\s+', ' ',x.split(' ',1)[1].strip()) ) \
        for x in f if x.split(' ',1)[0].strip() in trLst )
    trLbs={k: ' '.join([c if c!=' ' else '<space>' for c in v]) \
          for k,v in trLbs.items() }
with open("train_gt.txt",'w') as f:
    for k,v in trLbs.items(): f.write(k+' <space> '+v+' <space>\n')

# Creating valid_gt.txt file
with open("Transcripts.txt") as f, \
    open("Partitions/ValidationLines.lst") as g:
    trLst=set((x.strip()) for x in g)
    trLbs=dict( \
        ( x.split(' ',1)[0].strip(), \
          re.sub(r'\s+', ' ',x.split(' ',1)[1].strip()) ) \
        for x in f if x.split(' ',1)[0].strip() in trLst )
    trLbs={k: ' '.join([c if c!=' ' else '<space>' for c in v]) \
          for k,v in trLbs.items() }
with open("valid_gt.txt",'w') as f:
    for k,v in trLbs.items(): f.write(k+' <space> '+v+' <space>\n')

```

The produced file (for example `valid_gt.txt`) looks like:

```
# Format
# Line-ID          <char1> <char2> <char3> ...
...
035_322_001.r186 <space> S . <space> 2 <space>
035_323_001.r5   <space> 1 8 2 <space>
035_323_001.r205 <space> 3 <space>
035_323_001.r6   <space> C o n s t i t u t i o n a l <space> C o d e <space>
071_010_002.r47 <space> ( B ) <space> ( C ) <space>
...
```

To obtain the file `symbols.txt` with the list of symbols to be trained:

```
# Creating symbols.txt file
gawk '{for (i=2;i<=NF;i++) L[$i]=""}
      END{for (l in L) print l}' train_gt.txt valid_gt.txt |
sort |
gawk 'BEGIN{print "<ctc>\t0"}{print $1"\t"NR}' > symbols.txt
```

... and the Python script version:

```
# Creating symbols.txt file
with open("train_gt.txt") as f, open("valid_gt.txt") as g:
    s1=set(y for x in f for y in x.strip().split()[1:])
    s2=set(y for x in g for y in x.strip().split()[1:])
with open("symbols.txt", 'w') as f:
    f.write('<ctc> 0\n')
    for i,x in enumerate(sorted(set.union(s1,s2))):
        f.write(x+' '+str(i+1)+'\n')
```

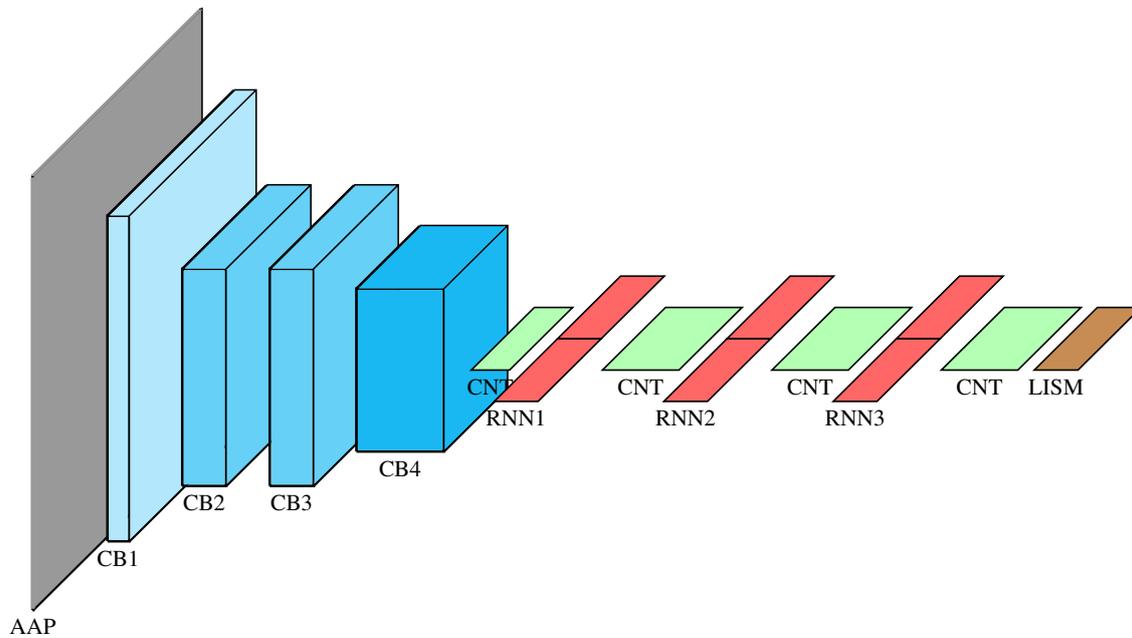
The content of the obtained file `symbols.txt` looks like:

```
# Format
# Symbol ID
<ctc>      0
!          1
"          2
#          3
&          4
...
0          14
1          15
2          16
3          17
...
<space>   27
...
A          31
B          32
C          33
D          34
...
```

WARNING: Note that “<ctc> 0” must be the first line of the list (excluding the comment lines). This is the special CTC symbol, whose ID is 0.

6 CRNN Training Phase

The CRNN topology and meta-parameters (number of convolutions and recurrent neural network layers, activation function types, drop-out parameter values, kernel size and stride steps, etc.) adopted for this practice session are shown in Fig. 4.



Reference Parameters:

AAP: Adaptive Average Pooling, **Output-Size:**16

CB1: **Cnn-Knl/Strd:** $3 \times 3 / 1 \times 1$, **Feat-Mps:**12, **B-Nrm:**Yes, **Act-F:**L-Relu, **MxPool:** 2×2 , **DrpO:**0

CB2: **Cnn-Knl/Strd:** $3 \times 3 / 1 \times 1$, **Feat-Mps:**24, **B-Nrm:**Yes, **Act-F:**L-Relu, **MxPool:** 2×2 , **DrpO:**0

CB3: **Cnn-Knl/Strd:** $3 \times 3 / 1 \times 1$, **Feat-Mps:**48, **B-Nrm:**Yes, **Act-F:**L-Relu, **MxPool:** 0×0 , **DrpO:**0

CB4: **Cnn-Knl/Strd:** $3 \times 3 / 1 \times 1$, **Feat-Mps:**48, **B-Nrm:**Yes, **Act-F:**L-Relu, **MxPool:** 2×2 , **DrpO:**0

RNN1: **Rnn-type:**BLSTM, **Units:**256+256, **DrpO:**0.5

RNN2: **Rnn-type:**BLSTM, **Units:**256+256, **DrpO:**0.5

RNN3: **Rnn-type:**BLSTM, **Units:**256+256, **DrpO:**0.5

LISM: Linear+Softmax layer, **DrpO:**0.5

CNT: Concat layer, after the CB4, concatenates all Feat-Mps from all pixels of an individual column into a single vector.

Figure 4: The adopted CRNN topology consists of 4 *Convolutional Blocks* (CB), followed by 3 *Recurrent Neural Network* layers (RNN) and a full connected *Linear+Softmax* layer (LISM). References: Kernel/Stride (Knl/Str), Feature Maps (Feat_Mps), Batch Normalization (B-Nrm), Activation Function (Act-F), Max-Pooling (MxPool), Drop-Out (DrpO).

6.1 Creating a CRNN Model

Before starting the training process, **PyLaia** requires a file containing the specifications of the CRNN topology to be trained. To this end, we create a `model` directory where the file containing such topology specifications (also named `model`) and also the files with the proper trained parameters (weights of the network) will be stored. Then, we proceed to generate the `model/model` file, defining the topology according to Fig. 4, by running **pylaia-htr-create-model** command with the adequate options as follows:

```
# Generate a topology model/model
mkdir model
pylaia-htr-create-model \
  --logging.overwrite True \
  --logging.level INFO \
  --logging.to_stderr_level INFO \
  --logging.filepath train-crnn.log \
  --common.train_path ./model/ \
  --common.model_filename model \
  --fixed_input_height 0 \
  --adaptive_pooling avgpool-16 \
  --crnn.cnn_kernel_size [3,3,3,3] \
  --crnn.cnn_stride [1,1,1,1] \
  --crnn.cnn_dilation [1,1,1,1] \
  --crnn.cnn_num_features [12,24,48,48] \
  --crnn.cnn_batchnorm [True,True,True,True] \
  --crnn.cnn_activation [LeakyReLU,LeakyReLU,LeakyReLU,LeakyReLU] \
  --crnn.cnn_dropout [0,0,0,0] \
  --crnn.cnn_poolsize [2,2,0,2] \
  --crnn.use_masks True \
  --crnn.rnn_type LSTM \
  --crnn.rnn_layers 3 \
  --crnn.rnn_units 256 \
  --crnn.rnn_dropout 0.5 \
  --crnn.lin_dropout 0.5 \
  symbols.txt
```

The main settings of this command are:

- directory path where to store the file with topology definition and files with trained parameters.
- name of the file with topology definition.
- adaptive average pooling to input line images independently of their height.
- convolutional block hyper-parameters: number of convolutional layers, number of feature maps, whether or not to apply batch normalization, type of activation function, kernel size and stride, drop-out value, max-pooling kernel size per layer.
- recurrent NN block: number of RNN layers, type of RNN, number of RNN units per layer, drop-out value.
- file containing the list of symbols to be trained.

6.2 Training CRNN

To launch of the training process itself, we resort to the **PyLaia** command: **pylaia-htr-train-ctc**. This is in charge to train the CRNN model applying a gradient descend technique called *Back-Propagation Through Time* (BPTT).

```
# Train the CRNN model
pylaia-htr-train-ctc \
  --logging.overwrite False \
  --logging.level INFO \
  --logging.to_stderr_level INFO \
  --logging.filepath train-crnn.log \
  --common.train_path ./model \
  --common.model_filename model \
  --data.batch_size 32 \
  --data.color_mode L \
  --optimizer.name Adam \
  --optimizer.learning_rate 0.0003 \
  --train.augment_training True \
  --train.delimiters ["<space>"] \
  --train.early_stopping_patience 20 \
  --trainer.gpus 1 \
  --trainer.auto_select_gpus True \
  symbols.txt ["Lines/"] train_gt.txt valid_gt.txt
```

The main settings of this command are:

- learning rate value.
- mini batch size and number of image channels to consider.
- whether or not to apply data augmentation.
- directory containing the line images and files with the list of IDs of training and validation samples respectively.
- list of symbols/chars to train.
- maximum number of consecutive epochs without a new lowest validation CER.

To check the usage of the NVIDIA® GPU with this task, employ the command **nvidia-smi** (it is assumed that package **nvidia-smi** has been installed in the local system).

WARNING: This training process takes a long time (around 8 hours depending of the GPU). To skip this step, download the model already trained from:

```
# Download the file into the $HOME/WorkDir directory
[ -d $HOME/WorkDir/model ] && rm -rf $HOME/WorkDir/model
wget --no-check-certificate \
  https://www.prhlt.upv.es/~ahector/BERN/Aux/model-CRNN_bentham.tgz
# Decompress it into the $HOME/WorkDir directory
tar xzf model-CRNN_bentham.tgz -C $HOME/WorkDir/
```

7 Decoding and Evaluation

Once the training process of a CRNN model has finished, this is ready to be used in the HTR decoding process of new given line images. This decoding process is actually carried out by the **PyLaia** command **pylaia-htr-decode-ctc** with adequate options as shown below:

```
# Decoding de validation partition
pylaia-htr-decode-ctc \
  --logging.level NOTSET \
  --logging.to_stderr_level INFO \
  --common.train_path ./model \
  --data.batch_size 32 \
  --data.color_mode L \
  --decode.include_img_ids True \
  --decode.separator " " \
  --decode.use_symbols True \
  --trainer.gpus 1 \
  --trainer.auto_select_gpus True \
  --img_dirs ["Lines/"] \
  symbols.txt Partitions/TestLines.lst > hypotheses.txt
```

The main settings of this command are:

- path of the directory containing trained model parameters.
- mini batch size and number of image channels.
- path of the directory containing line images to decode.
- list of trained symbols/characters.
- file with the list of IDs of test samples to decode.

To check/see the first 10 lines of the produced `hypotheses.txt` file:

```
head hypotheses.txt
```

```
# Format
# Line-ID      <char1> <char2> <char3> ...
071_022_003.r221 <space> : e d <space> Y o u r s e l f . <space>
071_030_001.r41  <space> M u t i l a t i o n . <space>
071_030_001.r110 <space> E x p o s i t i o n <space>
071_022_003.r59  <space> t h e <space> t i m e . <space>
071_022_003.r248 <space> 2 3 <space>
071_030_001.138  <space> I I I <space>
071_030_001.140  <space> ( C ) <space>
071_030_001.131  <space> I I <space>
071_163_004.r117 <space> a r e <space> i n t r o d u c t o r y . <space>
071_030_001.r75  <space> M a i n - <space> T e a t . <space>
...
```

7.1 Evaluation of the CRNN Decoding Output

For evaluating decoding outputs, the reference ground-truth file `test_gt.txt` is required, which has to be generated from files `Partitions/testLines.lst` and `Transcripts.txt` in the same way as it was done for generating `train_gt.txt` and `valid_gt.txt`.

To evaluate the quality of the HTR decoding output in the file `hypotheses.out`, the figures *Character Error Rate* (CER) and *Word Error Rate* (WER) are computed. The following shell script carry out that through the command `compute-wer`, which belongs to the Kaldi Toolkit:¹⁰

```
# Compute CER without initial and final "<space>" symbols
compute-wer --mode=strict \
  ark:<(gawk '{ $2=""; $NF=""; print}' test_gt.txt) \
  ark:<(gawk '{ $2=""; $NF=""; print}' hypotheses.txt) |
grep WER | sed -r 's|%WER|%CER|g'

# Compute WER
compute-wer --mode=strict \
  ark:<(gawk '{ printf $1; \
    for (i=2;i<=NF;i++)
      if ($i=="<space>") printf " "; else printf $i;
    print ""
  }' test_gt.txt) \
  ark:<(gawk '{ printf $1; \
    for (i=2;i<=NF;i++)
      if ($i=="<space>") printf " "; else printf $i;
    print ""
  }' hypotheses.txt) |
grep WER
```

This is the Python script version for computing CER and WER:

```
# Require "editdistance" package
# pip install editdistance
import editdistance
with open("test_gt.txt") as f, open("hypotheses.txt") as g:
    ref=dict( \
        (l.split(' ',1)[0], l.split(' ',1)[1].split()[1:-1]) for l in f )
    hyp=dict( \
        (l.split(' ',1)[0], l.split(' ',1)[1].split()[1:-1]) for l in g )

# Compute CER without initial and final "<space>" symbols
rc=[ref[k] for k in ref.keys()]
hc=[hyp[k] if k in hyp else '' for k in ref.keys()]
numedit = sum([editdistance.eval(x,y) for x,y in zip(rc,hc)])
numWords = sum([len(ref[k]) for k in ref.keys()])
print("CER(%%) = %6.2f [ %d / %d ]" % \
      (numedit/numWords*100,numedit,numWords))

# Compute WER
rw1=[ ''.join(x).replace('<space>',' ').strip().split() for x in rc ]
hw1=[ ''.join(x).replace('<space>',' ').strip().split() for x in hc ]
numedit = sum([editdistance.eval(x,y) for x,y in zip(rw1,hw1)])
numWords = sum([len(k) for k in rw1])
print("WER(%%) = %6.2f [ %d / %d ]" % \
      (numedit/numWords*100,numedit,numWords))
```

¹⁰<http://kaldi-asr.org>

8 Decoding Setup to get also Information about Word Locations

Basically, to get also the segmentation information at word (or character) level in the decoding process, we use the option: `--decode.segmentation` as shown below:

```
# Decoding de validation partition
pylaia-htr-decode-ctc \
  --logging.level NOTSET \
  --logging.to_stderr_level INFO \
  --common.train_path ./model \
  --data.batch_size 32 \
  --data.color_mode L \
  --decode.include_img_ids True \
  --decode.separator " " \
  --decode.use_symbols True \
  --decode.segmentation word \
  --trainer.gpus 1 \
  --trainer.auto_select_gpus True \
  --img_dirs ["Lines/"] \
  symbols.txt Partitions/TestLines.lst > wordSeg_info.txt
```

To have fancy visualization of predicted transcripts along with their segmented words on corresponding line images, we make use of the Python script `visualize_segmentation.py`:

```
# Download Python script "visualize_segmentation.py" for displaying lines
with predicted word segmentation
wget --no-check-certificate https://www.prhlt.upv.es/~ahector/AERFAI/
PyLaia/Aux/visualize_segmentation.py
```

Run the Python script on the first 10 predicted lines

```
# Remove "OutSeg" directory in case it already exists
rm -rf OutSeg
# Run the script on the first 10 predicted lines and save created images
on "OutSeg" directory
./visualize_segmentation.py Lines OutSeg <(head -10 wordSeg_info.txt)
```

Fig. 5 shows some line images with their predicted texts and marked word segmentations.

9 Proposed Exercise

By using only the 30% of the available training samples, train and evaluate the decoding output of a new CRNN whose topology is defined in Fig. 6.

References

- [1] Martin Maarand, Yngvil Beyer, Andre Kåsen, Knut T. Fosside, and Christopher Kermorvant. A comprehensive comparison of open-source libraries for handwritten text recognition in norwegian. In Seiichi Uchida, Elisa Barney, and Véronique Eglin, editors, *Document Analysis Systems*, pages 399–413, Cham, 2022. Springer International Publishing.

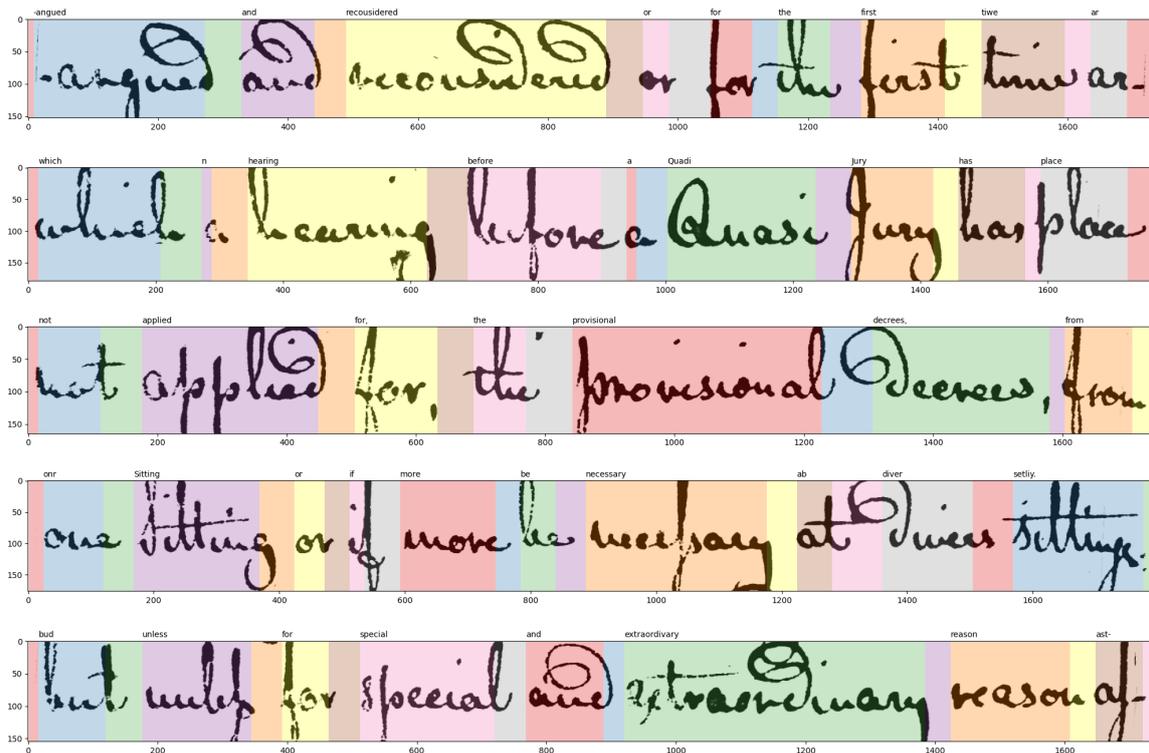
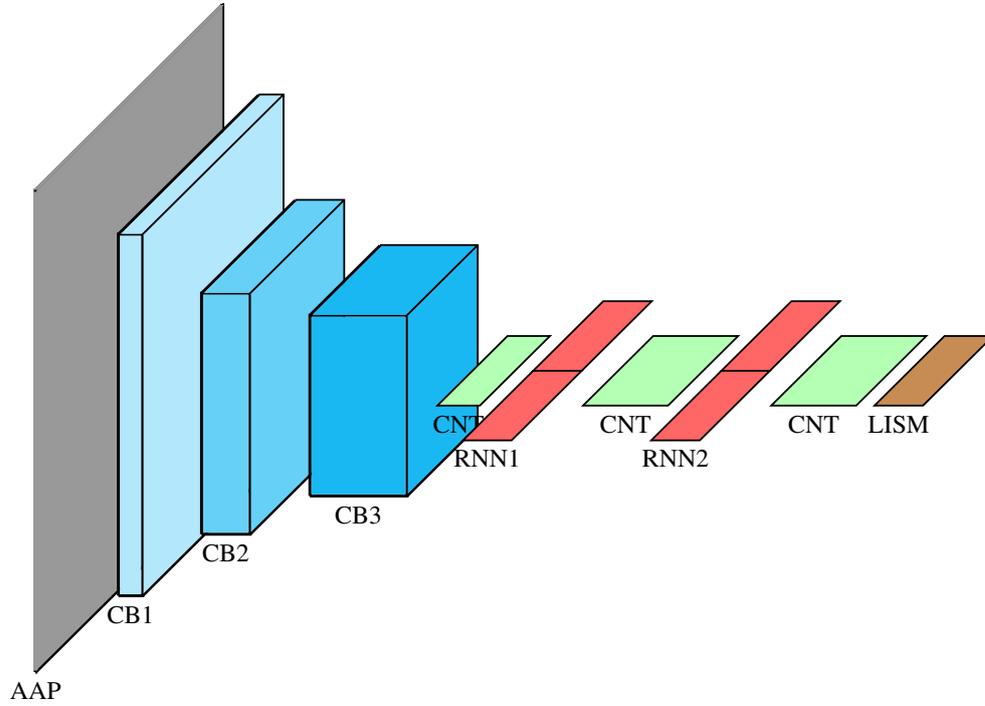


Figure 5: Examples of predicted texts along with their word segmentations displayed on corresponding line images.

- [2] S. Pletschacher and A. Antonacopoulos. The PAGE (Page Analysis and Ground-truth Elements) format framework. In *International Conference on Pattern Recognition*, pages 257–260, 2010.
- [3] J. A. Sánchez, V. Romero, A. H. Toselli, and E. Vidal. ICFHR2014 competition on handwritten text recognition on tranScriptorium datasets (HTRtS). In *International Conference on Frontiers in Handwriting Recognition*, pages 181–186, 2014.
- [4] Joan Andreu Sánchez, Verónica Romero, Alejandro H. Toselli, Mauricio Villegas, and Enrique Vidal. A set of benchmarks for handwritten text recognition on historical documents. *Pattern Recognition*, 94:122 – 134, 2019.
- [5] M. Villegas, V. Romero, and J. A. Sánchez. On the modification of binarization algorithms to retain grayscale information for handwritten text recognition. In R. Paredes, J.S. Cardoso, and X.M. Pardo, editors, *Pattern Recognition and Image Analysis: 7th Iberian Conference, IbPRIA 2015, Santiago de Compostela, Spain, June 17-19, 2015, Proceedings*, pages 208–215, 2015.



Reference Parameters:

AAP: Adaptive Average Pooling, **Output-Size:**16

CB1: **Cnn-Knl/Strd:** $3 \times 3 / 1 \times 1$, **Feat-Mps:**12, **B-Nrm:**Yes, **Act-F:**L-Relu, **MxPool:** 2×2 , **DrpO:**0

CB2: **Cnn-Knl/Strd:** $3 \times 3 / 1 \times 1$, **Feat-Mps:**24, **B-Nrm:**Yes, **Act-F:**L-Relu, **MxPool:** 2×2 , **DrpO:**0

CB3: **Cnn-Knl/Strd:** $3 \times 3 / 1 \times 1$, **Feat-Mps:**48, **B-Nrm:**Yes, **Act-F:**L-Relu, **MxPool:** 2×2 , **DrpO:**0.5

RNN1: **Rnn-type:**BLSTM, **Units:**128+128, **DrpO:**0.5

RNN2: **Rnn-type:**BLSTM, **Units:**128+128, **DrpO:**0.5

LISM: Linear+Softmax layer, **DrpO:**0.5

CNT: Concat layer, after the CB3, concatenates all Feat-Mps from all pixels of an individual column into a single vector.

Figure 6: The adopted CRNN topology consists of 3 *Convolutional Blocks* (CB), followed by 2 *Recurrent Neural Network* layers (RNN) and a full connected *Linear+Softmax* layer (LISM). References: Kernel/Stride (Knl/Str), Feature Maps (Feat_Mps), Batch Normalization (B-Nrm), Activation Function (Act-F), Max-Pooling (MxPool), Drop-Out (DrpO).