Contents

Project Machine Learning
— Milestone 1 —

Safaa Alnabulsi, Abd Almuhsen Allahham, Yamen Ali

November 24, 2019

1 Overview

Optical Character Recognition (OCR) is the process of converting scanned images of handwritten, typed or printed text into recognizable electronic text. Despite of the excellent progress made in OCR filed nowadays, it is still a challenging task to get a pre-trained model that is applicable to a wide range of printed typographies, fonts, scripts and publication periods.

In their paper, Shafait et al.[1] introduced a 1D bidirectional LSTM architecture with CTC algorithm for performing OCR tasks. They also showed a performance evaluation for their approach on both printed English recognition and Fraktur (a common historical German script) recognition. It is worth mentioning that the aforementioned approach, unlike other contemporary systems, doesn't employ any post-processing or language modelling techniques.

2 Introduction

Many printed OCR algorithms depend on segmenting the input text line images into characters and then applying a classifier on those segmentations. This approach requires a complex and reliable design of a segmentation methods since an error in segmentation would lead to an error in the final classifier output.

However, unsegmented OCR approaches, were no segmentation of the text line images is needed, have been applied successfully as well. HMM is probably the most common among these approaches. Although HMM avoids many difficulties of segmentation-based OCR systems, nevertheless a considerable effort still has to be invested in building the HMM itself.

With the recent advancements in machine learning and neural networks field it was possible to perform printed OCR tasks using RNN and LSTM architectures without the need for a segmentation step.

Following, we try to summarize the main aspects of Shafait et al.[1] approach for printed OCR:

- **Model:** The suggested model is a 1D bidirectional LSTM neural network. The model input is a sequence of text line images frames. No feature engineering is needed, so the input is just the images pixels. In other words, each text line image will be converted into a sequence of frames and fed to the BLSTM. It is important to note that each frame may or may not contain a character since there is no segmentation process involved.

  The BLSTM output at each time step represents the character probability associated with the input frame at the same time step. But since there was no segmentation applied on the input, an alignment is needed between the BLSTM output and the actual targets (i.e. text transcriptions). This alignment is achieved by applying a CTC algorithm on the model output.

- **Connectionist Temporal Classification (CTC):** It is basically an output layer/algorithm designed for sequence labelling tasks. We will omit technical details here, but it is worth mentioning that CTC outputs a probability distribution over label sequences, this distribution is going to be used in the objective function that drives the whole model optimization.

3 Dataset Overview

The used dataset is called *GT4HistOCR* (German and Latin ground truth for historical OCR)[6]. The dataset consists of printed text line images paired with their transcription (see Figure1). In total, the dataset consists of 313,173 line pairs from books printed in Fraktur types between 15th and 19th century. The dataset is divided into five subcorpora. Each subcorpora consists of multiple books.

Table 1 shows an overview of the subcorpora. For each subcorpora, we indicate the name, number of books, the printing period, the number of line text images and the language:

Subcorpus	Books	Period	Lines	language
Reference Corpus ENHG	9	1476-1499	24,766	ger
Kallimachos	9	1487-1509	20,929	ger,lat
Early Modern Latin	12	1471-1686	10,288	lat
RIDGES Fraktur	20	1487-1870	13,248	ger
DTA19	39	1797-1898	243,942	ger

Table 1: GT4HistOCR overview

The corpora's transcription was done manually and later checked and corrected by trained philologists. The corporas are not constructed according to language or period but rather their material has been merged and harmonized.

Jnd des hertzongen myt fcharper wer

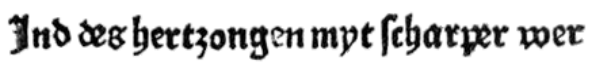

Figure 1: A sample from GT4HistOCR data *describes*. The data comes in pairs, At the bottom we see the text line image. At the top we see the corresponding text.

3.1 Dataset Directory Structure

The dataset is structured in the following manner. There are five directories correspond to the five subcorpora. Each subcorpora directory contains multiple directories correspond to the books that belongs to the subcorpora. Each book directory is named with publishing year and book title. The book contains the text line images and their transcriptions as text files.

```
dataset/
├── EarlyModernLatin
│   ├── 1471-Orthographia-Tortellius
│   │   ├── 00001.bin.png
│   │   ├── 00001.gt.txt
│   │   ├── 00002.bin.png
│   │   └── 00002.gt.txt
│   ├── 1471-Orthographia-Tortellius
│   ├── 1476-SpeculumNaturale-Beauvais
│   └── 1483-Decades-Biondo
├── Kallimachos
├── RIDGES-Fraktur
├── RefCorpus-ENHG-Incunabula
└── dta19
```

The mapping between text line images and their transcription is based on the same name. For instance, the transcription of text line image *(EarlyModernLatin/1471-Orthographia-Tortellius/00001.nrm.png)* can be found in text file *(EarlyModernLatin/1471-Orthographia-Tortellius/00001.gt.txt)*
.

3.2 Text Line Images

In general, no special aspects can be obtained from the text line images in the dataset since they have been already annotated and verified by experts. The only remark here is that the images are not all of the same size, so a resizing or padding is needed in order to get them all at the same size. Figure 2 and Figure 3 shows the max and min text line images in the dataset:

Figure 2: Max Sample size:(4505, 181), Subcorpus: EarlyModernLatin, Book: 1483-Decades-Biondo, Image Name: 00407.bin.png.

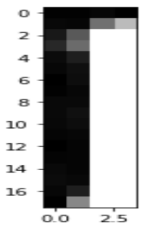

Figure 3: Min Sample size:(4, 18),Subcorpus: dta19, Book: 1852-storm-gedichte, Image Name: 00566.bin.png. The sample represent the number (1)

3.3 Text Transcriptions

In order to get more insights about the data, we came up with some statistics that we think it could help us with the upcoming milestones. Table 2 shows the alphabet size of each dataset.

Subcorpora	Alphabet Size (Unique Characters)
Reference Corpus ENHG	91
Kallimachos	103
Early Modern Latin	180
RIDGES Fraktur	145
DTA19	183

Table 2: Alphapet size of Datasets

To visualize those insights we used histogram to plot the frequency of top 20 unique characters occurrences in a each data set. It is apparent that the most common characters in all datasets are almost the same. This insights may become helpful later on for validating the correctness of the model. (see Figure 4)

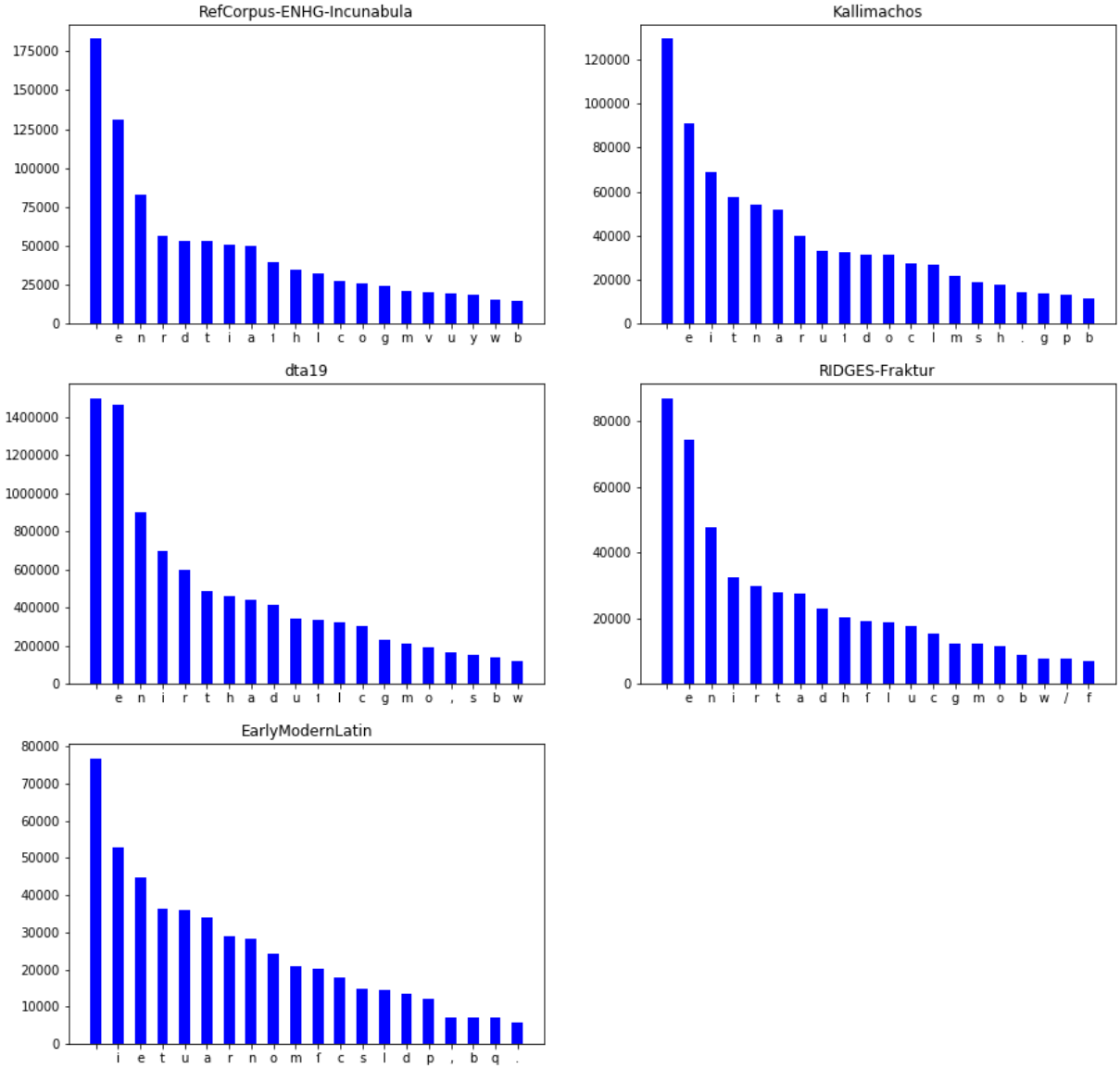

Figure 4: Top 20 frequented characters histogram in GT4HistOCR

3.4 Features and Transformations

No feature extraction is performed on text line images. The images will be read as black and white pixels (i.e. 0s and 1s) and fed to the LSTM. However, couple of transformations are performed on the images before sending them through the model.

- **Resize/Padding:** all images should be resized to same size. We think Resizing the images up is better than down in order not to lose any information.
- **Rotate:** we rotate the images 90 degrees in order to split each image into multiple frames later on during training. (i.e. each frame represents a time step for the LSTM)

In their paper, Shafait et al. [1] perform a normalization step before the training. The line size normalization procedure is rather complex and it aims for rescaling the text line images in a way that fixes the possible curves in their baseline and x-heights. We won't be performing such normalization step since GT4Hist dataset doesn't contain curved text line images.

4 Baseline Model And Evaluation

OCR is inherently a challenging problem, hence building a model that can be considered as a good baseline for the printed Latin and Fraktur OCR is a non trivial task. Following, we describe an approach for designing a baseline model that can be used on text line images without the need for segmentation.

Furthermore, there exist open source OCR engines that are easy and straightforward to use. The most famous ones are (Tesseract[2], OCRopus[3]) and both of them were used as baselines in Shafait et al paper[1]. Tesseract offers a segmented approach for performing OCR tasks, therefore we can also use its results as baselines for the unsegmented BLSTM approach.

4.1 Evaluation Criteria

Before discussing baselines, we have to introduce an evaluation function to assess the quality of a model predictions. In their paper, Shafait et al. [1] used the ratio of insertions, deletions and substitutions relative to the length of the ground-truth text. Hence, the error rate function used for evaluation can be defined as:

$$Err = \frac{Levenshtein(gt\ text, predicted\ text)}{len(gt\ text)} \tag{1}$$

Where $Levenshtein(gt\ text, predicted\ text)$ is the levenshtein distance between ground-truth text and the predicted text.

Although it is possible to use other metrics like (character accuracy or word accuracy), Levenshtein distance introduces more reasonable way to validate the model output. Table 3 shows couple of examples where the previously proposed error rate function gives more rational values especially in cases where the difference between the ground truth text and the predicted text is hard even for humans to detect (e.g.: ' J ' considered ' I ' or ' ſ ' considered ' f ').

Ground Truth	Model Output	Word Accuracy	Char Accuracy	Err
Jnd des hertzongen	Ind des hertzongen	70%	5.5%	0.05
Gud gereytfcafft	Gud gereytfcafft	50%	18.7 %	0.18

Table 3: Comparison between different validation approaches

4.2 Baseline Model Concept

As a baseline model we implemented a CNN (Convolutional Neural Network) and trained it on single-character images of size (70, 23). To evaluate the baseline model against text line images, the next approach was followed:

- Use a moving window to extract frames from the text line image. Each frame is of the size (70, 23) which means a character should fit inside it.
- Slide the moving window one pixel at a time.
- Feed the extracted frames into the CNN, the result will be a probability vector for each frame.
- Select the char with highest probability from each of the probability vectors.
- Remove all chars with probability under a certain threshold.

- Join the remaining chars together and remove the consecutive occurrences of the same char.

We consider the final output from the previous algorithm as the predicted text, and we apply the evaluation criteria discussed in 4.1.

In order to train the model we need images of characters and their associated labels. This is not possible to obtain from GT4HistOCR dataset since we have only text line images with no segmentation info. Hence, a synthetic data was used for the training and evaluation. Two types of of images were generated:

- **Single-character images:** these images contain only one character from the GT4HistOCR alphabet and their width and height are selected to be just enough to contain a single character.
- **Text line images:** we got the text from the GT4HistOCR dataset and generated text line images using the same font that was used in generating the single-character images.

In order to evaluate the baseline model on the original GT4HistOCR text line images, we would need to use a suitable *Fraktur* font type for the single-character images generation. We tried few online available *Fraktur* font types, but the results were not good enough to be considered as baseline results.

Thus, we just went with generating the text line images ourselves. The content of the synthetic text line image was obtained from GT4HistOCR, but the font used was standard *Arial*.

4.3 Baseline Model Architecture

The CNN we built has the following specifications:

- **Layers:** one convolution layer and one output layer.
- **Input:** the input of the CNN is a single-character image.
- **Output:** the output is a vector with the same size as our alphabet (i.e. after applying a soft max function on this vector it will be become the probability vector of the alphabet).
- **Filters:** the number of used filters is equal to the number of characters in our alphabet.
- **Kernel:** the kernel size is equal to generated single-character images width and height.

4.4 Baseline Model Results

We trained five models (one for each subcorpora in GT4HistOCR). The only difference between models is the alphabet used for generating the single-character images used in training. We have also trained an extra model on standard English alphabet (i.e. A-Z, a-z). For each model evaluation, we picked 50 samples from each book within its corresponding subcorpora, then generated text line images for the samples text using standard Arial font. Table 4 shows the results of this evaluation approach.

Subcorpus	Samples Number	Mean Error	Mean Error(English Model)
Reference Corpus ENHG	450	0.414	0.362
Kallimachos	450	0.601	0.390
Early Modern Latin	600	0.868	0.410
RIDGES Fraktur	1000	0.945	0.382
DTA19	1950	0.806	0.398

Table 4: Mean error rate produced by baseline models predictions on each dataset

It can be noticed that standard English model gives the best results. This is mainly because *Fraktur* and *Latin* have many special glyphs, and during evaluation when we slide the window across the text line image, many frames will contain different parts of neighbouring characters which may look like an actual Latin or Fraktur glyph.

The followed approach apparently has downsides and the biggest one is the fact that we trained and evaluated our model on synthetic data.

4.5 Tesseract Results

We used **Tesseract(version 4)**[2] and ran it with mode –**oem 0** in order to use the legacy engine(i.e. the engine which doesn't depend on LSTM). From GT4HistOCR dataset (the original not the synthetic) we picked up to 51 sample from each book in each subcorpora. Table 5 shows the results of evaluating Tesseract output using the previous char error rate function (mentioned in 4.1).

Subcorpus	Samples Number	Mean Error	Model
Reference Corpus ENHG	450	0.480	frk
Kallimachos	450	0.4104	lat
Early Modern Latin	600	0.254	lat
RIDGES Fraktur	1000	0.280	frk
DTA19	1950	0.281	frk

Table 5: Mean error rate produced by Tesseract Legacy engine predictions on each dataset. The model column indicates the pre-trained Tesseract language model that was used in the experiment

4.6 Overfitting

In general since the main building block in the model we are implementing is BLSTM, then we may come across an overfitting issue. Some standard techniques can help avoiding this, such as :

- Plot training, test and validation errors in order to detect overfitting when it occurs .
- Use drop out techniques.
- Early stopping for the learning process.
- Select a proper hidden layer size and hidden layers number (huge networks will overfit if the data is not enough).

We can also benefit from the way GT4HistOCR dataset is structured. When we are training on one subcorpora, we can pick some books as a hold out set, train on the remaining books and use the hold out set to do validation. By following this way we would ensure no overfitting is happening since the model is validated on entire books it had never seen before.

5 Discussion

5.1 The Challenges of OCR for Historical Printings

- **historical typographies:** there are lots of different printing types and high variability in letter shapes, arrangement, appearance. Same applies for numbers and symbols.

Figure 5: Two samples from the Dataset "RefCorpus-ENHG-Incunabula", book "1476-Historij-Wierstaat". The historical fonts can be noticed along with the different appaerance of the same letter

- **special glyphs:** the same glyph must have the same transcription, even if the glyph has different context dependent meanings. However, that's not the case with historical typographies and old fonts which sometimes show the same glyph for the letter which lead to confusion of the machine model and let it randomly output one of the different characters or character sequences it has learnt to associate with the glyph. For instance, most *Fraktur* fonts do not differentiate between the alphabetic characters I and J and use the same glyph for both. Also with the long "s" which is recognized as "f" letter.

Pontanus: Progymnasmata Latinitatis (1589)

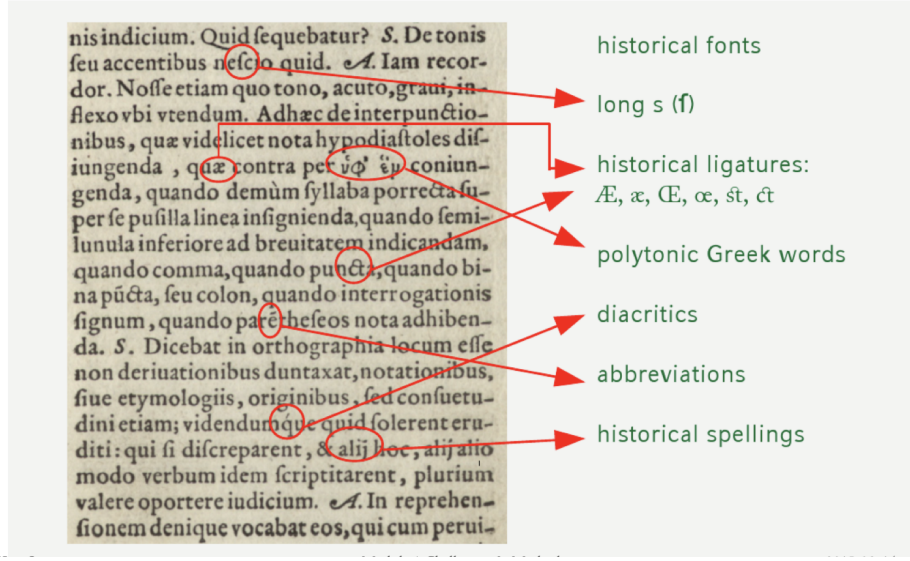

Figure 6: Overview of some special glyphs of one page from Progymnasmata Latinitatis (1589) book, EarlyModernLatin dataset[5]

- **historical fonts and historical spellings:** high variability in spelling and morphology which leads to increase in OCR error rate.
- **incunabula:** or sometimes incunabulum, is a book printed in Europe before the year 1501. An incunabulum printing has special abbreviation signs.
- **different transcription guidelines:** when pooling data from different corporas, which have been transcribed by different transcription guidelines, the guidelines have to be inspected in order to regularize different data sets to a common norm.

- **unbalanced categories** the data samples/targets are originated from textbooks. Thus it has some lexical model that is used by the language, and there usually we have unbalanced usage of characters, i.e. some letters are used commonly, whereas some other letters are rarely used (which can be seen clearly in the histograms in Figure 4). This will lead to a problem where some rarely used letters could be missing in the training set. Consequently, it will lead to a faulty model that cannot recognise those rare letters. The solution would be in implementing a smart data loader that ensures both training and testing data have relatively enough occurrences of each letter.

- **different samples size**: (See Figure 2 and Figure 3). This requires preprocessing of samples before training.

6 Summary and Next Steps

For this milestone, we focused our efforts on understanding the *GT4HistOCR* dataset by visualizing its statistical aspects and checking its samples characteristics.

Moreover, we created a baseline model which depends on synthetic single-character images for training and syntehtic text line images with text from GT4HistOCR for evaluation.

As for building the model suggested by Shafait et al.[1], we created the initial OCR LSTM model and used CTC as the loss function. We already implemented forward and backward passes and started running the training on the cluster.

For the next milestones, we have the following in mind:

- Introduce a better approach for unifying the size of the text line images before passing them to the model.

- Enhance the data loading and (train - test) splitting algorithm in order to avoid unfair assignments of low frequency characters (i.e. Not to assign all occurrences of one character to test set without having any in training set)

- Currently we are using *Pytorch* implementation of *CTC*, but we would like to try out and compare with *Baidu*[4] implementation as it might have better performance.

References

[1] T. M. Breuel et al. "High-Performance OCR for Printed English and Fraktur Using LSTM Networks". In: *2013 12th International Conference on Document Analysis and Recognition*. Aug. 2013, pp. 683–687. DOI: 10.1109/ICDAR.2013.140.

[2] Open Source OCR Engine. *Tesseract OCR*. https://github.com/tesseract-ocr/tesseract.

[3] Open Source Project. *OCRopus*. https://en.wikipedia.org/wiki/OCRopus.

[4] Open Source Project. *warp-ctc*. https://github.com/baidu-research/warp-ctc.

[5] Uwe Springmann. *Workshop: OCR and postcorrection of early printings for digital humanities*. https://www.cis.uni-muenchen.de/ocrworkshop/data/slides/m1-challenges.pdf. Sept. 2015.

[6] Uwe Springmann et al. *Ground Truth for training OCR engines on historical documents in German Fraktur and Early Modern Latin*. 2018. arXiv: 1809.05501 [cs.CL].