Contents

Project Machine Learning
— Milestone 2 —

Safaa Alnabulsi, Abd Almuhsen Allahham, Yamen Ali

January 26, 2020

1 Recap Of Milestone 1

In the previous milestone, we worked mainly on two aspects: Dataset understanding and training the baseline model. To understand the *GT4HistOCR* dataset, we visualized its statistical aspects and checked its samples characteristics.

We created the baseline model and generated synthetic single-character images for training and syntehtic text line images with text from *GT4HistOCR* for evaluation.

Moreover, we also started with building the initial *OCR LSTM* model, used *CTC* as the loss function, implemented forward and backward passes and started running the training on the cluster.

2 Data preparation

2.1 Subcorpora Selection

As mentioned in the first report, the *GT4HistOCR* dataset consists of five subcorpora of printed text line images paired with their transcriptions as shown in Table 1 . For this milestone, we will concentrate our efforts on two subcorpora.

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 [2]

Our subcorpora of choice are the subcorpus **Reference Corpus ENHG** and the subcorpus **Early Modern Latin**.For all the subsequent tasks we will keep one book from each subcorpus as a hold out book, and we will train and test our models using the other books.

The intuition behind this approach is to evaluate how good the model is in performing OCR on text line images from a book that was never used in training.

We implemented our code to take the holdout book name as a parameter, so in principle any book can be chosen as a holdout. For consistency, we will always choose **1499-CronicaCoellen (1628 samples)** from **Reference Corpus ENHG** and **1483-Decades-Biondo (915 samples)** from **Early Modern Latin** as holdout books.

2.2 Text Line Images Preprocessing

Before passing the text line images to the model some transformations are needed to be performed in order to get them in the correct shape and form. We revisited our previous approach and came up with following transformations:

- **Fix Text Line Image Transformation:** The data set contains some irregular text line images where the image height is much bigger than the image width. To make sure such images won't affect our model, we implemented a simple transformation that would resize in an appropriate way to make their height smaller than their width (by padding the images from the side with white background).
- **Image Thumbnail Transformation:** This transformation is used to scale a text line image into a thumbnail (i.e. scale the text line image in away that preserves the aspect ratio in order to avoid any distortion). We apply this transformation to speed up the training process because smaller images means faster training. But since this is a down sampling transformation, the image quality will be affected which in turn will affect the model accuracy. So we have here a trade off between accuracy and speed.

  Note: This transformation is carried out using PIL library. In order to perform it, the thumbnail function expects a max size parameter. For the used subcorpus **Reference-Corpus-ENHG**, intuitively we chose $500 * 20$ as the max size, while in **Early Modern Latin** we selected and re-implemented the transformation in a way that the text line image height will be always 50 pixels. However, in order to search for the best values, the max image size has to become a hyperparameter and a tuning has to be carried out for it, which we decided not to do for this milestone.
- **Image Padding Transformation:** This transformation will pad all images to the same width and height such that they can be processed in batches. The unified height and width are the same as the max size used in the image thumbnail transformation.
- **Unfold Image Transformation:** This transformation will convert the text line image into frames (vertical cuts).

3 Methodology

As mentioned in the first report, the suggested method by Shafait et al.[1] depends on two main aspects: BLSTM neural network and CTC loss. Following we provide more in depth explanation for these methods.

3.1 Model Architecture

As shown in Figure 1 the used model consists of three layers in the following order:

1. BLSTM layer which consists in turn of number of BLSTM cells.
2. Dropout layer.
3. Fully connected linear layer followed by log softmax function.

The model output is followed by CTC loss function which is used as the objective of the back propagation algorithm.

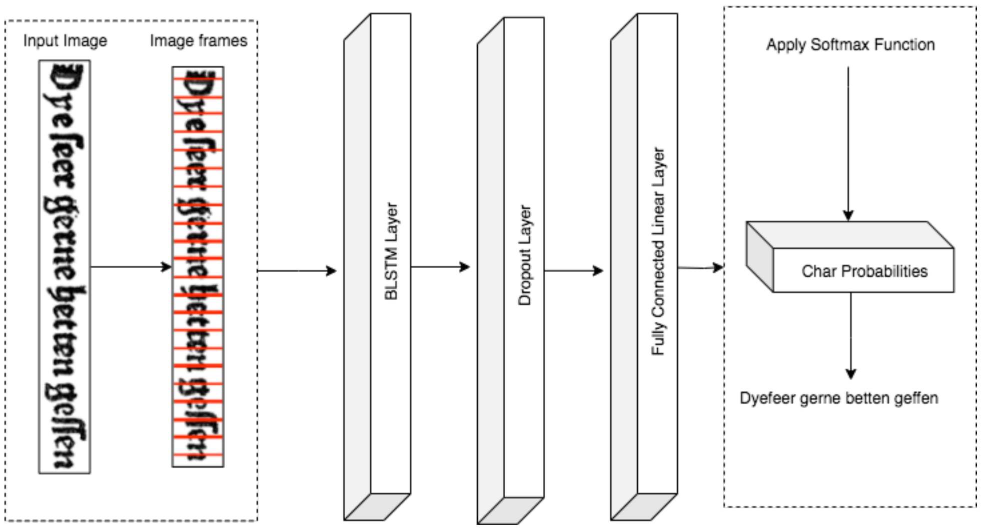


Figure 1: The proposed model architecture

3.2 BLSTM Layer

A 1D bidirectional LSTM neural network. The layer input is a sequence of text line images frames. As shown in Figure 2, each text line image is converted into consecutive frames which are fed to the model one by one (i.e. at each time step a frame is extracted from the image and fed to the model).

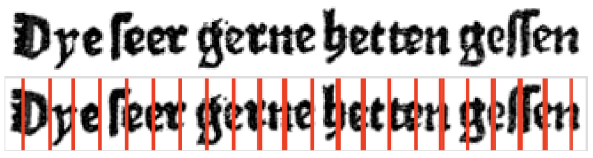


Figure 2: An example of extracting frames from a text line image

The output at each time step , after passing it through the fully connected layer and log softmax, represents the character probability associated with the input frame at the same time step.

ALPHABET	frame1 output	frame2 output	frame3 output	..
a	0.12	0.51	0.33	..
b	0.32	0.02	0.23	..
c	0.04	0.21	0.07	..
.	..	..	..	..

Table 2: Example of the model output.Each column represents the output at a certain time step

3.3 Connectionist Temporal Classification (CTC):

CTC is an algorithm that computes the probability distribution over label sequences. It doesn't require a predefined alignment between the input frames and the output labels, so no segmentation is needed for the text line images into character images.

The input of CTC is the probabilities associated with the frames (i.e.: the output of the log softmax function) and the output is the probability distribution of the correct frames labelling. This distribution can be used in a loss function in order to train the model.

It is worth mentioning that CTC loss is computed efficiently using dynamic programming approach (i.e. Viterbi algorithm). Figure 3 show how the CTC computes

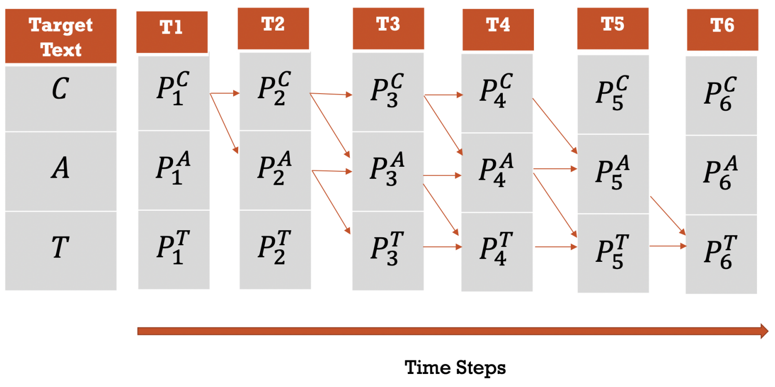

Target Text
T1
T2
T3
T4
T5
T6
C
A
T
P_1^C P_2^C P_3^C P_4^C P_5^C P_6^C
P_1^A P_2^A P_3^A P_4^A P_5^A P_6^A
P_1^T P_2^T P_3^T P_4^T P_5^T P_6^T
Time Steps

Figure 3: An example of computing the probability distribution of the correct frames labelling. P_1^C means the probability associated with the letter C in the model output at the first time step (i.e. the probability of the letter C in the model output when we fed the first frame for the model in the input). The arrows shows how we should calculate the joint probability distribution in away that guarantees a correct frames labelling.

the probability distributions for the correct labelling of 6 frames that are associated with the word (CAT).

Finally, since multiple frames can cover one letter in the text line image, we have to remove all occurrences of similar consecutive letters from the output of the model when we infer it. For example, a valid output of the model associated with Figure 3 would be (CCAATT). So to get the final prediction (CAT) we would need to remove all the duplicates.

3.4 Training and Evaluation

As stated in 2.1, we keep one book as a holdout data and train on the others. Additionally, we split the training data into (80% train data - 20% test data). The intuition behind such splits is to get a feeling how good the model is in transcribing:

- **Training Data**: Text line images the model was trained on.
- **Test Data**: Text line images the model was not trained on but they come from a book that was used in the training.
- **Holdout Data**: Text line images the model was not trained on and they come from unseen book.

3.4.1 Training algorithm:

Our training algorithm goes like the following :

1. Calculate max text length and alphabet from the selected data set.
2. Split dataset into (train, test) and holdout.
3. Shuffle (train, test) dataset and split it into (80% train data, 20% test data).
4. Apply transformations on the text line images.
5. Train the model using the train dataset.
6. Report the model error (i.e.: Levenshtein distance metric) on train, test and holdout data sets.

We depend on data loaders in our implementation, so splitting the data is being performed on indices level not on the actual text line images and the transformations are applied only when the loader loads the images required for the batch before passing it to the model (i.e.: lazy loading).

3.5 Hyperparameters

Following are the hyperparameters we pass for our training script. Later on we show in detail what values we tried and their respective results.

- **Frame Size:** The size of one frame extracted from the text line image (i.e.: The width of the frame). After applying the previously mentioned transformations, each text line image will be converted into frames, each frame is a vertical cut in the text line image with (Height = Thumbnail image height, Width = Frame size). This parameter value has to be chosen in a way that guarantees at most one character from the text line image will appear in the frame.
- **Hidden Layer Size:** The number of hidden units in the LSTM cell. This number will be implicitly duplicated since we have BLSTM.
- **Hidden Layers Number:** The number of LSTM cells to use in the model.
- **Dropout Ratio:** The drop out ratio used in the drop out layer in the model.
- **Learning Rate:** The learning rate used by SGD optimizer during the training.
- **Momentum:** The momentum used by SGD optimizer during the training.
- **Epochs:** The number of cycles used in training.
- **Batch Size:** The number of text line images fed to the model at once.
- **Clipping Value:** A value to clip the gradient at in order to avoid exploding gradient problem.

4 Empirical Results

4.1 Error Measure

As mentioned in the first report, we will use the following error function suggested by Shafait et al. [1] as an evaluation criteria.

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

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

4.2 Baseline vs OCR LSTM

As a recap of our baseline model concept, we generate char images and text line images using Arial font and train a CNN filter for each char image. Then for inference, we extract frames from the generated text line images, feed them to the CNN and take the highest probability as the prediction of the frame, finally we consider all probabilities above a certain threshold. For more details about this approach refer to **4.2.1 Baseline Model Concept** section in the previous report.

For this milestone, in order to compare the OCR LSTM model with the baseline model we took the following approach:

1. Generate char images from the **Reference Corpus ENHG** dataset alphabet using Arial font.
2. Train baseline model using the generated char images.
3. Create a new dataset called **Reference Corpus ENHG Synthetic**. We keep the same structure of the original dataset (i.e. books, text line images and transcriptions), but we generate the text line images using Arial font.
4. Train the OCR LSTM model on 8 books from **Reference Corpus ENHG Synthetic** and keep one book **1499-CronicaCoellen** as a holdout.

5. Evaluate and compare baseline model and OCR LSTM model performance on the holdout book using the error measure from 4.1.

For baseline model, we were able to get $error = 0.433$ on the holdout book after training for 1000 epochs. For OCR LSTM approach, we used cross validation on different model architectures to get a hint of what could work best for us. Then we trained 12 models for 2000 epochs each. Table 3 shows the best, median and worst obtained models.

Hidden Layer Size	Hidden Layers	Frame Size	Learning Rate	Error	Improvement
100	3	3	0.01	0.121	72.05 %
1500	2	5	0.01	0.339	21.70 %
100	1	5	0.01	0.703	-62.35 %

Table 3: Evaluation error obtained by training OCR LSTM models on synthetic dataset and the improvement achieved over the baseline model evaluation error

Generally speaking, it is noticeable that deeper models are more likely to achieve better error rates . These results are consistent with what cross validation suggested and also holds for original dataset as will be shown later on. Figure 4 shows an example of a text line image and the related prediction results.

owmatn demn begymnmnens eckdchemn ades gechet)j

owat in dem begynne eins iecklichen alders gefchiet fij.

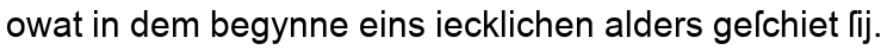

owat in dem begynne eins iecklichen alders geſchiet ſij.

Figure 4: An example of the baseline and the best OCR LSTM predictions. The top line is the baseline model prediction. The middle line is the OCR LSTM model prediction. The bottom line is the actual generated text line image. This example is not necessary the best we could find, it was just picked to fit in the page margin

For the median and worst cases, we can think of two reasons behind the results:

- **Frame Size:** Appropriate frame size selection is crucial because it would affect the difficulty of the problem. For instance, choosing (frame size = 3) will almost guarantee that each frame contains a part of a single char from the text line image. Whereas, choosing (frame size = 5) will cause some frames to contain parts of two consecutive chars from the text line image which in turn affects the probabilities output and the whole learning process.

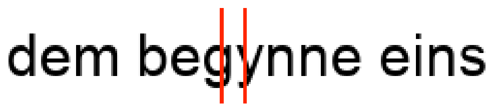


Figure 5: An example of frame size selection that might cause some frames to contain parts of two consecutive text line image chars

- **Model Architecture:** When comparing the (test and train) errors for the previous three models in Table 4, it is clear we have a case of overfitting for the median model and underfitting for worst model.

Hidden Layer Size	Hidden Layers	Train Error	Test Error	Evaluation Error
100	3	0.010	0.013	0.121
1500	2	0.005	0.076	0.339
100	1	0.682	0.663	0.703

Table 4: Train, test and evaluation errors for best, median and worst models

Hence, we can notice that Shafait et al. [1] approach is valid and it can achieve better results than the baseline model for simple synthetic fonts.

4.3 Results On Original Data

Here we use the same previously discussed aspects of Subcorpora selection, training algorithm, data splitting and evaluation. The only difference is that we train the OCR LSTM model using the original **GT4HistOCR** dataset not the synthetic one.

No baseline model evaluation were performed here due to the absence of a suitable font that can be used to generate the char images needed to train the baseline model.

As mentioned before, cross validation was used to get an intuition what architectures would work best for us, and hyperparameters tuning was used to run some experiments (we were not able to use it as our main algorithm to come up with proper hyperparameters values due to cluster performance issues that we will discuss later on).

4.3.1 Reference Corpus ENHG Results

Here we show subset of the trained models sorted by the error achieved on the holdout book. Table 5 shows the trained models sorted by the error achieved on the holdout book. The best, worst and median models are highlighted in gray.

ID	HL Size	HL number	Frame Size	Train Error	Test Error	Holdout Error	Batch Size	Dropout Ratio	Training Time(hrs)
1	200	3	1	0.006	0.016	0.054	400	0.1	19.8
2	1500	2	2	0.0001	0.022	0.092	100	0.3	234.8
3	100	3	2	0.064	0.064	0.127	800	0.3	20.13
4	200	3	2	0.072	0.072	0.147	400	0.1	13.6
5	1000	3	5	0.001	0.096	0.245	100	0.3	73.7
6	500	2	5	0.116	0.141	0.302	500	0.3	37
7	1500	2	5	0.0	0.137	0.312	100	0.3	130.2
8	100	1	5	0.245	0.253	0.378	800	0.3	13.5

Table 5: Subset of trained models results. HL stands for (Hidden Layer). Train, Test, Holdout errors are the levenshtein errors on training, test and holdout data consecutively.

All the previous models where trained with **(Learning Rate = 0.01, Clipping Value = 1000, Epochs = 2000, Momentum = 0.9)**.

Figure 6 shows the results of inferencing the trained models using a text line image from the holdout book. We notice that the best model (i.e. second row from top) was able to convert the text line image to text very well although the text line image itself is kind of blurry and suffers from some pixels loss.

cleyder die velle van den beiſten.

cleyder die velle van denbeiſten.

ceyder die vele van denbeiſten.

cerder die vole van den beſten

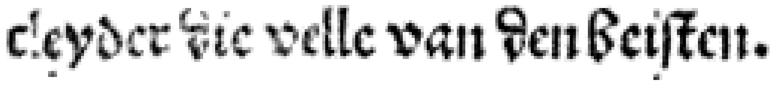
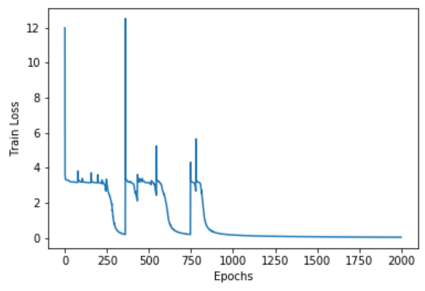
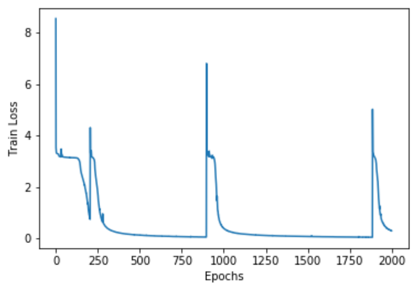

cleyder die velle van den beiſten.

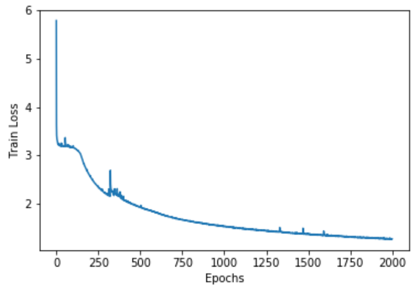
Figure 6: We see in order from top to bottom:the ground truth text, the best model result, the median model result, the worst model result, the text line image.



Best Model



Median Model



Worst Model

Figure 7: The training error plots for best, median and worst model

4.3.2 Early Modern Latin Results

For this subcorpus, we have some differences regrading the hyper parameters because here we tried our own implementation of Image Thumbnail transformation as mentioned in 4.1 . Table 6 shows the trained models sorted by the error achieved on the holdout book. The best, worst and median models are highlighted in gray.

ID	HL Size	HL number	Farme Size	Train Error	Test Error	Holdout Error	Training Time(hrs)
1	150	2	1	0.001	0.03	0.095	13.5
2	150	3	1	0.004	0.037	0.095	19.5
3	150	4	2	0.0001	0.028	0.102	16.2
4	150	2	2	0.0002	0.036	0.109	16.9
5	200	3	3	0.0001	0.037	0.121	8.2
6	100	4	4	0.002	0.041	0.127	5.9
7	200	2	4	0.00004	0.04	0.137	6.0
8	100	2	4	0.001	0.052	0.15	5.6

Table 6: Trained models results.HL stands for (Hidden Layer). Train, Test, Holdout errors are the levenshtein errors on training, test and holdout data consecutively.

All the previous models where trained with **(Learning Rate = 0.05, Epochs = 1000, Momentum = 0.9, Batch Size = 200, Dropout Ratio = 0.3, Image Height = 50 pixel)**. Figure 8 shows the results of inferencing the trained models using a text line image from the holdout book.

ſcus utraque: coniuncta claſſe: cõtra populoniæ

ſcis utraque: coniuncta claſſe: cõtra populoniæ

ſc is utraque: coniuneta claſſe: cõtra populoniæ

ſc:ais utraque: coniuncta cdaſſe: cõtra populoniæ

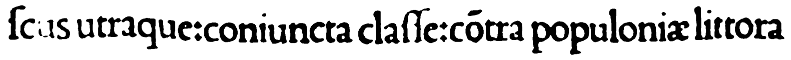


Figure 8: We see in order from top to bottom: the ground truth text, the best model result, the median model result, the worst model result, the text line image.

5 Discussion

5.1 Efficiency And Accuracy

From the previous explanation and the obtained results we can summarize the factors affecting the model accuracy and training time in the following three aspects:

- **Thumbnail Transformation**: Scaling down the text line images before feeding them to the model has great influence over the training performance because smaller images means:
  - Fewer steps to be made in the BLSTM layer.
  - Smaller memory footprint per text line image which allows us to increase the batch size. Bigger batches means more utilization for the GPU power; hence faster training.

  However, scaling the images down would mean lower resolution which will make the learning process harder and will affect the model accuracy eventually (i.e. will increase the Levenshtein error).
- **Model Architecture**: It can be noticed from Table 5 that training models with more hidden units need more time and might result in overfitting situation **(e.g. Model 7)**. On the other hand, models with not sufficient hidden units will finish the training faster but suffer from underfitting **(e.g. Model 8)**. Choosing a suitable model architecture will allow the model to generalize well and finish training in reasonable time **(e.g. Model 1)**
- **Frame Size:** A bigger frame size means the text line image will be divided into fewer frames, hence fewer steps are needed in the BLSTM layer and the training becomes faster. But as shown in Figure 5, a wide frame might contain parts of multiple chars in the text line image which will make the problem harder and will result in lower model accuracy.

In all the above aspects we have a trade off between training efficiency (i.e. GPU and memory usage) and model accuracy. Through intuition, cross validation and hyperparameters tuning we were able to reach a suitable setup that achieved good results even on the hold out book.

5.2 Retraining

Retraining the model when new data is available would be straight forward and the efficiency depends directly on the previously mentioned aspects. As an example, the best model **Model 1 (Reference Corpus ENHG)** needs around **0.614 seconds** to complete training on one batch (i.e. the time needed for the model to do forward and backward pass on 400 text line images). With such information we can form and idea about the time needed to train the model on the new data and judge the training efficiency.

5.3 Problems

Although increasing the batch size will speed up the training process, but bigger batches require more memory and this was one of the biggest issues we faced while training on the provided cluster because we were getting **(CUDA out of memory)** exceptions randomly and frequently, so we had to reduce the batch size significantly in order to make our jobs run.

The memory issues made it hard for us to use cross validation and hyperparameters tuning as our main techniques to reach the best possible results, so we ended up running cross validation for 100 epochs just to get an intuition what model architectures would work best for us.

5.4 Confidence Measure

As explained in Methodology 3 the model output for each frame is chars probabilities and the char we assign for the frame is simply the one with the max probability, hence we can consider the joint probability of assigned chars for all frames as a confidence measure for the model predictions (i.e. the multiplication of max char probabilities for all frames) which can be expressed as the following formula:

$$confidence = \prod_{i=1}^{\#frames} max(output_i) \tag{2}$$

6 Conclusion

The suggested methodology achieved really good results converting the text line images into text. The fact that no language model was needed and the training was relatively fast make this approach highly practical and once implemented easy to adapt to any other language or font. On the other hand, the BLSTM model and the CTC loss function are not easy to comprehend and work with, which might make the implementation of this approach rather a challenging task in the absence of suitable framework support and tools.

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] 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]`.