Contents

Project Machine Learning
— Milestone 3 —

Safaa Alnabulsi, Abd Almuhsen Allahham, Yamen Ali

March 28, 2020

1 Recap Of Milestone 2

In the previous milestone, we explained in details the suggested model architecture and the connectionist temporal classification (CTC) loss function used for training.

We also introduced the idea of keeping one book as a holdout (i.e. never train the model on it) in order to use it later for evaluation. Moreover, we showed that the OCR LSTM model achieved better results than the CNN baseline model when evaluated on the holdout book of **Reference Corpus ENHG** subcropus.

Finally, we reported best, median and worst models results for **Reference Corpus ENHG** and **Early Modern Latin** subcorpora alongside some examples that show how good the models were able to convert the text line images into text without the use of any language model.

2 Prediction Methodology

2.1 Blank Character

Before going through the steps required to convert a text line image into text using a trained OCR LSTM model, we will explain briefly the concept of the blank character needed by the suggested LSTM approach.

As mentioned in the previous report, multiple frames can cover one character in the text line image so we have to remove all occurrences of similar consecutive characters from the output of the model when we infer it. Nevertheless, we can directly notice an edge case when we have two consecutive similar characters in the target text. For example, let's assume that the target text contains the word **Apple**, then in the model output we would have something like **AApppppplleee**. After removing consecutive similar letters we end up with **Aple**.

To overcome this problem, we introduce a blank character (e.g. $) that doesn't originally exist in the alphabet. This blank character will be added to the alphabet, moreover it will be added before and after each character in the target text. Then the word **Apple** in our example becomes **$A$p$p$l$e$** and the model output would be something like **$$AA$$ppp$$ppp$$ll$$eee$$$**. After removing the consecutive similar characters we get **$A$p$p$l$e$**. Finally, we just drop all the blank characters from the output and we get **Apple**.

It is worth mentioning, that the Pytorch CTC implementation takes care of adding the blank character before and after each character in the target text, so we only need to add this character to the alphabet and point the CTC loss function to its index.

2.2 Final Method

In order to infer the model, we divide the text line images into frames before feeding them to the model. Thus, the model will output for each text line image frame a probabilities vector with length equal to the size of the alphabet. For example, if the alphabet size is 26 and we divided the text line image into 20 frames then the model output will be a matrix of size $20 * 26$. Then for each frame we pick the char corresponding to the highest probability and finally we remove all duplicates. So we can summarize the final prediction method with the following steps :

1. Apply transformations to the text line image before passing it to the model. In this step the text line image will be converted into frames. For more details refer to section 2.2 from the previous report.
2. In the model output each column represents a probability vector associated with a frame. From each column, select the index with the highest probability. Hence, the selected indices in this step would be equal to the text line image frames number.
3. Convert the selected indices into the corresponding chars.
4. For all consecutive occurrences of the same char, keep only one and remove the rest. This is because multiple consecutive frames can be parts of the same char from the text line image.
5. Finally, remove all occurrences of the blank character $.

2.3 Confidence Measurement

2.3.1 Entropy

The information entropy, often just entropy, is a basic quantity in information theory associated to any random variable. It can be interpreted as the average level of "information", "surprise" or "uncertainty" inherent in the variable's possible outcomes. In other words, for a probability distribution (of a discrete random variable), the entropy is calculated with the following formula:

$$H(X) = \sum_{i=1}^{n} P(x_i) \log_b P(x_i)$$

where b is the base of the used logarithm.

Hence, the bigger the entropy is, the more uncertain we are about the outcome of this probability distribution and vice versa.

2.3.2 Entropy As A Confidence Measurement

The output of our suggested model as mentioned in 2.2 is a sequence of probability distributions that corresponds to the input frames. Moreover, each distribution represents how likely the input frame belongs to each character in the alphabet.

To get a confidence measurement on frame level, we start by calculating the entropy of the distribution associated with each frame. Then, since the entropy represents the uncertainty, we can simply use $1 - entropy$ as a confidence measurement.

Selecting a proper base during the entropy calculations (which is $\#alphabet\_characters$ in our case) will force the results to be in a range $[0, 1]$.

2.3.3 Model Confidence

After assigning a confidence for each frame we need a way to combine these confidences into a single value that represents the model confidence in its predicted text. In order to do that, we will introduce the notation of *local confidence*. The local confidence can be one of two types:

- **Frame Confidence** : Here we use the calculated confidence for each frame as it is.
- **Character Confidence**: We know that each predicted character could be represented with one or more frames, that means we can aggregate those frames together and calculate only one confidence that corresponds to one character.

  We can calculate the character confidence using one of the following three approaches:

  - ***Max*:** The character confidence is the maximum confidence of the frames that represent this character. In other words, this is the optimistic perspective to determine the character's confidence.

– ***Min*:** The character confidence is the minimum confidence of the frames that represent this character. In other words, this is the pessimistic perspective to determine the character's confidence.

– ***Mean*:** The character confidence is the average confidence of the frames that represent this character. In other words, this method tries to reduce the effect of both highly biased ***Max&Min*** methods.

The final model confidence is then calculated from the local confidences using one of the following approaches :

– ***Max*:** Here we select the maximum local confidence in the sequence. We think this method is invalid since in all of the tested cases we got some local confidences with extremely high values $\approx$ 1, especially the local confidences corresponding to the blank character.

– ***Min*:** Here we select the minimum local confidence in the sequence. Although this might be valid approach, we think it will be unjust to judge the model based on one frame or character prediction.

– ***Mean*:** Here we average over all local confidences in the sequence. This method tries to overcome the problems of the highly biased ***Max&Min*** methods. But we might still have a biased result here, that is the result of the high local confidences related to the blank characters in the sequence. A suggested solution here would be to ignore the blank character confidences in our calculations, since it was introduced by the learning algorithm and it almost has no meaning in the predicted text.

Table 1, shows the model confidence range (i.e. min confidence and max confidence) for the best model predictions on the holdout book of the **Reference Corpus ENHG** subcorpus. All of these confidences are calculated after removing the frames related to the blank character.

Model	Local Confidence			
Confidence	Frame	Char Min	Char Max	Char Average
Min	[0.32, 0.93]	[0.32, 0.93]	[0.32, 0.99]	[0.32, 0.93]
Max	[0.98, 0.99]	[0.98, 0.99]	[0.98, 0.99]	[0.98, 0.99]
Average	[0.74, 0.99]	[0.72, 0.98]	[0.73, 0.99]	**[0.72, 0.99]**

Table 1: The confidences range for all different combinations of local and model confidence calculations. These results are reported for the best model predictions on the holdout book from **Reference Corpus ENHG** subcorpus

Table 2, shows the model confidence range (i.e. min confidence and max confidence) for the best model predictions on the holdout book of the **Early Modern Latin** subcorpus. All of these confidences are calculated after removing the frames related to the blank character.

Model	Local Confidence			
Confidence	Frame	Char Min	Char Max	Char Average
Min	[0.24, 0.88]	[0.24, 0.88]	[0.24, 0.99]	[0.24, 0.93]
Max	[0.67, 1.0]	[0.67, 0.99]	[0.67, 1.0]	[0.67, 0.99]
Average	[0.67, 0.98]	[0.67, 0.98]	[0.67, 0.99]	**[0.67, 0.99]**

Table 2: The confidences range for all different combinations of local and model confidence calculations. These results are reported for the best model predictions on the holdout book from **Early Modern Latin** subcorpus

It is clear that the confidence of only one character's frames can make the final result highly biased if we took the **Min or Max** approaches to calculate the final model confidence. On the other hand, taking the average of local confidences, regardless of how they are calculated, will yield more natural model confidence.

To conclude, we will use the **Char Average** as the local confidence and then we will average all the local confidences to get the model confidence on the predicted text.

3 Model Explanation

The explanation of a model is a quantitative entity that gives us some sense of why the model has produced a particular output. Moreover, the explanation we are seeking should be in the same space as the input. Here we will discuss two approaches that we have checked alongside some pros and cons for of each them.

3.1 Layer-Wise Relevance Propagation (LRP)

This approach is relatively new and is considered one of the best. It takes the output and back propagate it to the input layer using some rules related to "Deep Taylor Decomposition"

- Pros
  - The explanations are clear and easy to understand
  - The Execution time is relatively fast
  - No sampling or training is needed to generate the explanation
- Cons
  - Not easy to implement for all different types of machine learning approaches. Here we faced some technical issues that we will mention later on.
  - Hyper-parameters tuning is needed (e.g. α, β, γ in γ-rule ...etc.) and this tuning process is time consuming.

3.2 Sensitivity Analysis (Gradient Analysis)

In this old approach the explanation is represented by the gradient of the input in each dimension. This gradient is calculated by back propagating the output to the input layer similarly to the error back propagation that is usually applied during the training phase.

- Pros
  - No need to any extra implementation, it works out of the box in most of machine learning frameworks.
  - Well studied analytically (even though the results are not always quite convincing)
- Cons
  - The generated explanations are very noisy due to the fact that the gradient is very sensitive to small changes of input values which means that any tiny noise or change on the input will be reflected dramatically on the explanation.

3.3 The Applied Approach

As Mentioned previously, the LRP would fit to our mission better but unfortunately we faced some issues implementing it. We tried to implement the approach provided in [1, see attached github sample], but that did not work out because they have their own customized implementation of LSTM while we used the standard Pytorch implementation. Hence, going with this approach would require re-implementing and retraining our models from scratch.

So after checking with the class supervisor, it was decided to go with the gradient approach, and here you can find an example of the result.

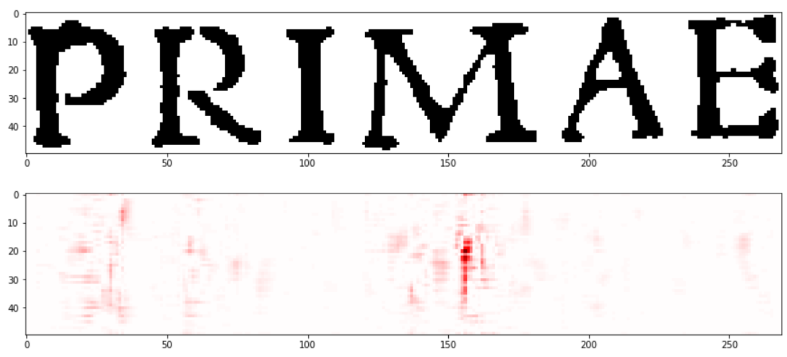

Figure 1: an example of one of the explanations applied on **Early Modern Latin** subcorpus.

We notice in Figure 1 that the explanation is not really informative and that's in our opinion due to:

1. The nature of the gradient.
2. The model itself, because the recurrent nature of the LSTM gives some frames higher importance than others since some decisions will be taken after evaluating multiple steps from the sequence. This could be noticed in the frames around 150-160 in the figure.

4 Evaluation

A reasonable evaluation metric of the model performance is the Levenshtein distance metric which was discussed and reported in the previous milestone. Nevertheless, here we will show a different approach to use when evaluating the models on character level instead of the final predicted text level.

4.1 The Confusion Matrix

If we consider each character in the alphabet as a class, then we can say that our model is performing a classification on frame level (i.e. each input frame will be classified into one of the classes/characters). The problem here is that we don't know what is the correct class for each frame because there is no pre-segmentation involved and the frames are aligned with the characters by the CTC algorithm. Hence, instead of looking into the classification on frame level we will assume it happens on character level by considering the predicted text characters as the predicted classes and the true text characters as the target classes. By doing so, we can easily build a confusion matrix by going through our predictions and counting the classification and misclassification on character level. For example, if the true text is **This is a cat** and the predicted text is **That is a cat**, then a proper confusion matrix would be the following:

	a	c	h	i	s	t
a	2	0	0	0	0	0
c	0	1	0	0	0	0
h	0	0	1	0	0	0
i	1	0	0	1	0	0
s	0	0	0	0	1	1
t	0	0	0	0	0	2

Table 3: Example of the confusion matrix built on character level. As usual the diagonal of this matrix indicates the correct classifications.

We can then compute the (Accuracy, Recall, Precision) for each character based on this matrix. Since the used alphabet is big we will not show the confusion matrix, but we will report some of the good and bad characters in terms of the previously mentioned metrics. As usual, these results are reported for the model predictions on the holdout books.

	Accuracy	Precision	Recall
¶	1.00	0.85	0.71
f	0.99	0.69	0.69
l	0.96	0.66	0.64
M	1.00	0.18	0.56
N	1.00	0.25	0.47

Table 4: Some metrics obtained from the confusion matrix for the predictions on the holdout book of **Reference Corpus ENHG** subcorpus

Since we are calculating the confusion matrix by doing one to one mapping between the true and predicted text, clearly there will be cases where a lot of characters are considered as misclassified just because there is a shift in the predicted text. Going back to the previous example, if our model predicts **Thiis is a cat** then the accuracy, precision and recall for the classes will go down although the text prediction itself is really good and informative. Following we will discuss two approaches to overcome this problem.

4.1.1 The Simple Approach

Here we will just ignore all predictions that have different length from the true texts. Table 5, shows the results for the characters reported in Table 4 after taking the ignore approach while building the confusion matrix.

	Accuracy	Precision	Recall
¶	1.00	0.96	0.79
f	1.0	0.88	0.88
l	0.99	0.86	0.84
M	1.00	0.25	0.91
N	1.00	0.53	0.75

Table 5: The metrics obtained from the confusion matrix for the predictions on the holdout book of **Reference Corpus ENHG** subcorpus after ignoring predictions with different length than true texts

We can notice from the obtained results that the model mainly suffers when it tries to predict characters which their shapes are close to each others (e.g. M & N). On the other hand, the model predicts with high accuracy and precision the characters which have unique shapes (e.g. ¶). Table 6, shows the confidence related with each char from Table 5. We notice that these confidences confirm our observations that the model suffers more when trying to predict similar looking chars.

	Confidence
¶	0.90
f	0.97
l	0.98
M	0.77
N	0.72

Table 6: The average confidence associated with each char across the predict texts from the holdout book of **Reference Corpus ENHG**

4.1.2 The Levenshtein Distance Approach

Here we will use the Levenshtein distance and the blank character to build a confusion matrix that takes into account strings with different lengths. The main idea here is to manipulate both the true text and the predicted text to have the same lengths.

We have 3 types of errors when it comes to text matching:

- **Insertion of extra character:** the model predicts an extra letter that doesn't exist in the true text. In order to fix the alignment, we insert the blank character in the true text which will map to the extra character in the predicted text.For example if the true text is **Apple** and the model returns **Applie**, we update the true text to be **Appl$e**
- **Deletion of a character:** the model misses a character which is supposed to show up in the prediction. In order to fix the alignment, we insert the blank character in the predicted text which will map to the missed character from the true text. For example if the true text is **Apple** and the model returns **Aple**, we update the predicted text to be **Ap$le**
- **Wrongly classified character:** we do nothing since the output of such misclassification will keep the lengths of both texts the same.

Applying these 3 rules, with the support of Levenshtein Algorithm, we will end up with two character series that have always the same length. Then we can easily build the confusion matrix based on these results, keeping in mind that the values we will get for the blank character **$** will have no specific meaning since the blank character was ignored from the model output and used here for alignment reasons.

	Accuracy	Precision	Recall
b	1.00	0.99	0.98
a	1.0	0.98	0.97
F	1.99	0.67	0.67
S	1.00	0.55	0.90
z	1.00	0.14	0.33

Table 7: Some metrics obtained from the confusion matrix for the predictions on the holdout book of **Early Modern Latin** subcorpus after applying the special mapping approach on the results

Table 8, shows the confidence related with each char from Table 7.

	Confidence
b	0.97
a	0.98
F	0.87
S	0.86
z	0.61

Table 8: The average confidence associated with each char from the predict texts from the holdout book of **Early Modern Latin**

4.2 Expected Time Saving

Let us assume that a human needs 5 seconds on average to perform the prediction of one text line image with 0% error and that computing a prediction using the OCR LSTM model takes 0 seconds. In order to calculate the time saving in different situations, we will need to define what does an error mean in the context of our OCR task.

Usually we consider a model prediction as an error if it is not equal to the target. We argue that using such notion in terms of this OCR task would be extreme especially taking into account that the texts are hard to read even for humans. For example if we consider only predictions that match the targets as correct, then the model error rate on the holdout book of **Reference Corpus ENHG** subcorpus is $\approx 89\%$. But if we consider the samples within $Levenshtein\ Error = 0.02$ to be also correct, then the model error rate drops to 76% and so on. The $Levenshtein\ Error$ is the error measurement we suggested in the last report and it has the following formula:

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

To calculate the time savings we use the following three simple formulas:

$$human\_effort = \frac{max(model\_error - acceptable\_error, 0) * \#samples}{100} \tag{2}$$

$$human\_time = human\_effort * human\_time\_per\_sample \tag{3}$$

$$time\_saving = \frac{(only\_human\_time - human\_time) * 100}{only\_human\_time} \tag{4}$$

Basically we are just calculating how much samples the human would need to do for the given acceptable error and model error. For example, if the model error is 9% and the acceptable error is 5% then the human would need to do 4% of the wrong samples.

Figure 2, shows the time savings for different model error rates and acceptable error rates. For instance, the red line corresponds to the time savings (in percentage) when the acceptable error rates ranges from 0% to 20% for a model with error rate = 33.05%, we get this model error rate when we consider all samples with $Levenshtein\ Error =< 0.06$ as correctly predicted samples.

By comparing Figures 2 & 3 , we can notice that the time savings for the **Early Modern Latin** subcorpus is lower due to higher model error rate in general.

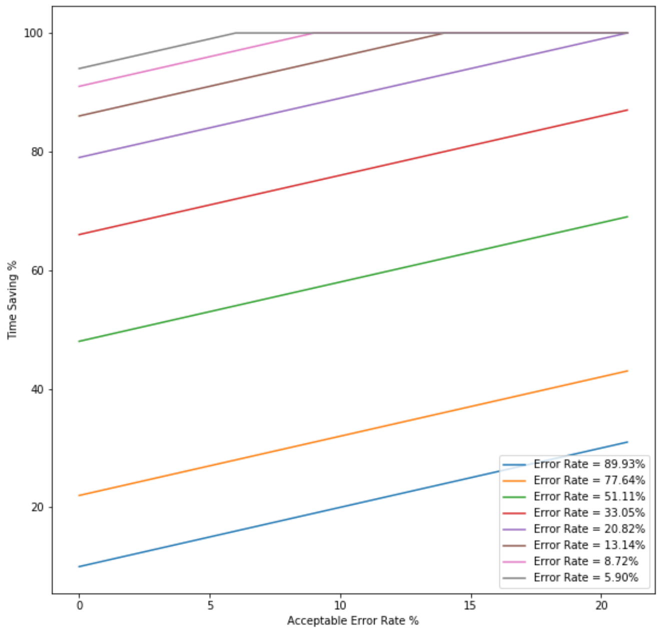

Figure 2: The time savings in predicting the holdout book samples of **Reference Corpus ENHG** subcorpus.

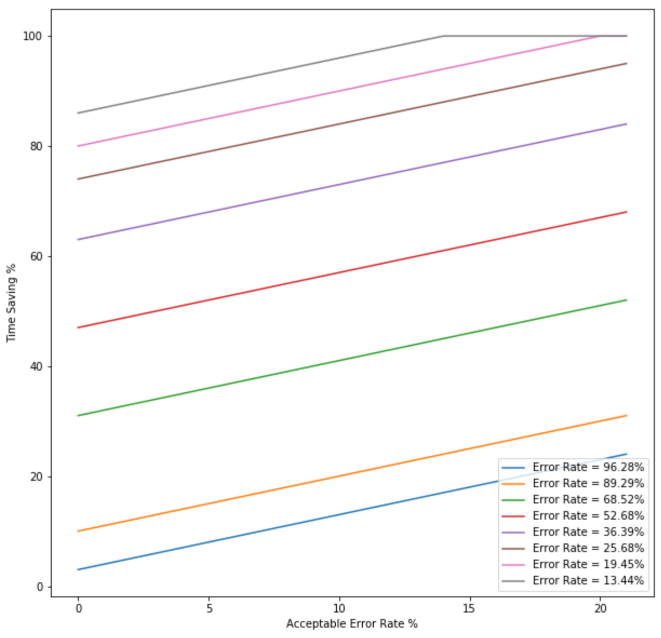

Figure 3: The time savings in predicting the holdout book samples of **Early Modern Latin** subcorpus.

4.3 Human And Machine Intelligence

Let's assume we have a combination of human and machine intelligence. When the confidence associated with the text is below a threshold, the machine will defer the decision to the human. Such decision can be explained by showing the confidences associated with each character in the predicted text. Figure 4, shows an example of the model prediction where the confidence associated with the predicted text was 0.84.

Anno dmfi. MCCCCxci.

Rnns důl. ChiC¶Cyll.

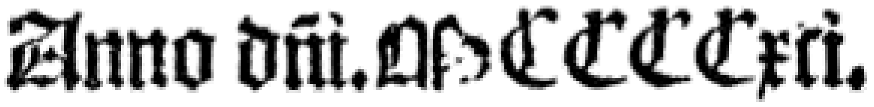
Anno dñi. MCCCCxci.

Figure 4: An example of a text predicted with low confidence.We see in order from top to bottom: the ground truth text, the predicted text, the text line image.

The associated confidence with each character in the predicted text is the following :
R: 0.50 **n:** 0.99 **n:** 1.00 **s:** 0.98 **__ :** 1.00 **d:** 1.00 **ů:** 0.70 **l:** 0.97 **.:** 0.99 **__:** 0.97 **C:** 0.49 **h:** 0.51 **i:** 0.76 **C:** 0.89 **¶:** 0.73 **C:** 0.74 **y:** 0.79 **l:** 0.96 **l:** 0.93 **.:** 0.99

We notice that the characters that are predicted correctly have high confidence (e.g. n, d, __). On the other hand, characters predicted wrongly have low confidence (e.g. R, h). So the human can easily understand why the model gives low confidence for the text and where it suffers by looking into the characters confidence.

Nevertheless, we can see some characters with high confidence although they were wrongly predicted. For example the last *i* was predicted as *l* with a relatively high confidence 0.93. One possible reason for this is that the model is slightly biased towards predicting the character *l* against *i* when there is a confusion. This can be seen by checking the confusion matrix entries for those characters :

$$confusion[i][l] = 15$$

$$confusion[l][i] = 5$$

These values indicate that the model predicted i as l three times more than it predicted l as i for the holdout book. Hence, we can say that the confusion matrix for the characters can give us an idea about the model bias for the characters with similar shapes.

Finally, it is obvious that some wrongly predicted characters with high confidence can't be attributed to a model bias (e.g. o predicted as s). Such errors can be attributed to the difficulty of the problem and to the fact that ,due to LSTM usage, some errors in the previous frames might have an effect on the future ones.

5 Outlook

One of the most obvious problems in the suggested methodology is the potential class imbalance. If we consider each character in the alphabet as a class, then the characters with low frequency in the training data will end up having a higher error rate. Introducing a weighted version of the CTC loss function can help in reducing the class imbalance effect which will lead to higher accuracy in the model predictions in general.

Another challenge related to this approach is the need of transcribing the text line images manually in order to get proper training data. Such task can be challenging and time consuming especially it may require experts to do it like in German frakture and early modern Latin transcription. As a suggested enhancement, we can generate synthetic data (i.e. text line images and their corresponding transcriptions) from the dataset alphabet in case we are able to create a computer font that is close enough to the actual one.

All in all, our results show that the claim of the original paper was correct and we were surprised by the quality of the transcriptions despite of the challenging task. The ability of the LSTM neural networks to capture the context and build a probability distribution for the characters without an involvement of any language model is really outstanding.

Finally, the fact that the best models were relatively small and didn't require a lot of training makes this approach really efficient and practical. Hence, we can write the required code once and reuse it ,out of the box, for any OCR task on any language given that the training data is in the proper structure.

References

[1] Leila Arras et al. "Explaining and Interpreting LSTMs". In: *Explainable AI: Interpreting, Explaining and Visualizing Deep Learning*. Ed. by Wojciech Samek et al. Cham: Springer International Publishing, 2019, pp. 211–238. ISBN: 978-3-030-28954-6. DOI: 10.1007/978-3-030-28954-6_11. URL: https://doi.org/10.1007/978-3-030-28954-6_11.