DeepLearn 2024: NLPRenaissance report

Forming the training and evaluation data

We changed the pipeline for dataset creation in three ways: Image extraction, line segmentation and the training/validation split.

Image extraction

In the provided colab notebook, the images of each page are extracted with PyMuPDF. This method reinterpolates the pages which can lead to a loss in resolution. To avoid this problem, we used `pdfimages` from `poppler-utils` to extract the image files directly from the PDF. We also processed the images with deskew to straighten the text lines horizontally. We provide the code for this here

Issues with handout word segmentation

In the sample notebook code, the extracted word images are saved as <transcription>.png. This means that for words that occur more than once, only the last image is saved. We know that word distribution in natural languages loosely follows Zipf's law. That is: the most frequent word is twice as frequent as the second most frequent, three times as frequent as the third most frequent etc. In the transcriptions, there are 327 words that appear more than once, with a total count of 2614, which means that 2287 word-image pairs are lost in the original data.

Another issue with the word segmentation in the handout code is that the number of words and number of bounding boxes for each line would often not be the same. Of the 609 lines in the training data, 192 of them (32%) had more words in the transcriptions than the number of bounding boxes from that line. There were also 75 lines where there were more bounding boxes than words. In the original code the words and images are matched up from the beginning and whichever sequence is longer is cut off at the length of the shorter one. We saw multiple cases where the word images did not have the correct transcription in the handout data.

Line segmentation

As mentioned above, there are issues with matching the correct word-image pairs in the handout code. And several of the algorithms we wanted to test are created for line-level data. Thus, we wanted to evaluate whether we could improve performance using line-level instead of word-level data. To create a line-level dataset, we first use a pre-trained Doc-UFCN model released by Teklia. We chose this model as it is trained on historical document datasets and is available on huggingface.

When inspecting the outputs of the Doc-UFCN model, we noticed that it sometimes splits a line into several horizontally and sometimes combines lines vertically. It also struggled to separate the content in the sidebar. One way to alleviate these problems is to fine-tune the model on our data, but this approach is not feasible within this hackathon since we don't have access to line annotations for the scanned pages in our dataset. Instead, we

used image processing techniques to clean up the line masks provided by the Doc-UFCN model.
To exclude the text from the sidebar, we segmented the image along the x-axis by thresholding the sum of each image column. This approach will remove text areas that are smaller than the main text area, effectively removing the text lines in the sidebar. To combine lines that were split horizontally, we used a morphological operator with a length-5 vertical structural element, and to separate lines that were joined vertically, we used a morphological opening operator with a length-201 horizontal structural element. Figure TODO shows a segmentation mask before and after this post-processing.

Finally, we used heuristics based on the bounding box locations to exclude headers and footers. The code uses hugging face for DoC-UFCN model inference and openCV for image processing, and it is available in the repository.

As the training and validation split in the sample notebook was made on word level, we had to create a new line-level training/validation split. We created a validation set of four pages (two spreads) and used the remaining 21 pages (10 spreads plus the first page) for the training set. To compare both word- and line-level segmentation approaches, we also created a word-level dataset with CRAFT using the provided code with two modifications: First, we modified the handout code to use unique identifiers as filenames to save the word images, such that words that occur multiple times don’t get overwritten. This led to a doubling in the amount of training data Second, we used the same pages for training and validation as for the line-level dataset to compare the approaches on the same validation data.

Models

We trained and/or fine-tuned three different model architectures

Handout model

We ran the model provided in the handout notebook on the handout data.

Tesseract

Tesseract is an open source OCR framework by Google, with pretrained models for multiple languages. They have a high-level and relatively easy-to-use repo for fine-tuning their OCR models in the tesstrain repository. We trained tesseract models on the word-level and line-level data with and without the Spanish base model from the tessdata_best repository.

TrOCR

Li et al. introduced the TrOCR model in TrOCR: Transformer-based Optical Character Recognition with Pre-trained Models . It consists of an image Transformer encoder and an autoregressive text Transformer decoder.

For this work, we used the trocr-large-printed and the trocr-large-spanish and fine-tuned on our line-level and word-level datasets. The code for this (which also shows all the parameters used for training is available here.

Post-processing

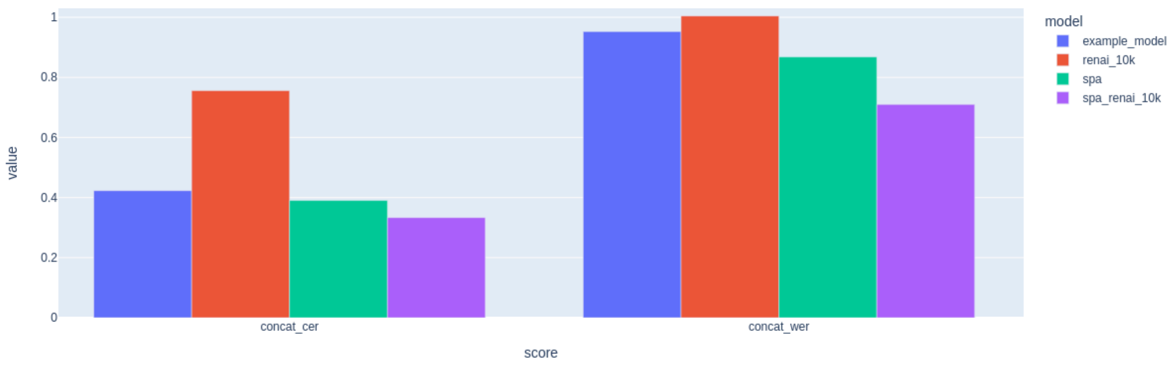
Because the transcriptions contain known irregularities, we evaluated whether the performance improves by post-processing the model output with rule-based post-processing. The post-processing works by following the rules in the provided PowerPoint and looking for matches in a dictionary. We used the unique words in the training data transcriptions to create the dictionary and added the ten thousand most common words from the Spanish Billion Words Corpus.

In our experiments, we compared both full post-processing (character-level substitution and dictionary lookup) and only dictionary lookup. The details and code for this post-processing are available in the repository.

Evaluation and model selection

Original word level data

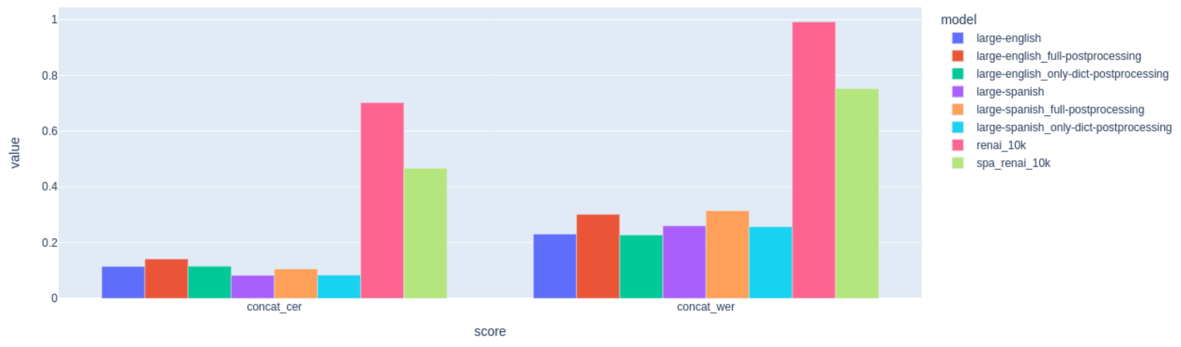
We first trained tesseract models (with and without the Spanish base model) on original data. We calculated the word error rate (WER) and character error rate (CER) on the validation set for our trained models, as well as the handout model and the pretrained Spanish tesseract model.



We observed that the tesseract model trained from scratch (renai_10k) performed worse than the handout model. The Spanish base model (spa) performed better than the handout model (example_model), and the fine-tuned model with the Spanish base (spa_renai_10k) performed the best on the handout data (0.333 CER and 0.710 WER).

Modified word level data

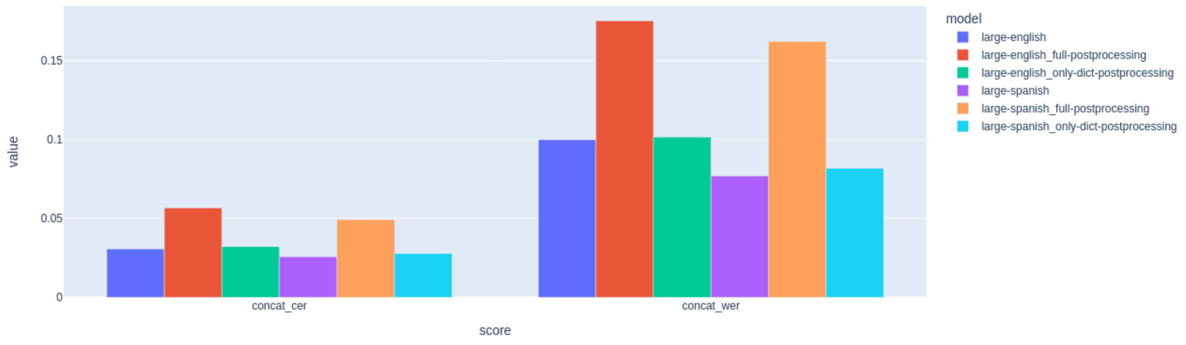
We fine-tuned both English and Spanish trOCR models, as well as tesseract models on the modified word level data, and evaluated on the validation set. All trOCR models greatly outperformed the tesseract models.



We also saw that the full post-processing in general worsened the model performance. The best model with regard to CER was the fine-tuned Spanish model without post-processing (large-spanish, 0.083 CER). The best model with regard to WER was the English model with dictionary lookup (large-english_only-dict-postprocessing, 0.227 WER).

Line level data

We fine-tuned both English and Spanish TrOCR models, as well as tesseract models on the line level data. Again, the TrOCR models all greatly outperformed the tesseract models, so we omit these from the plot to reduce complexity.



For the line level data and models, the Spanish model without post-processing (large-spanish) achieved the lowest WER and CER on the validation set (0.026 CER and 0.077 WER).

Conclusion

We saw that for small data sets like in this task we get better performance when we fine-tune pretrained models rather than train from scratch. We also saw that adding rule-based post-processing based on the rules from the task repository worsened model performance. The overall best model was the Spanish TrOCR model fine-tuned on our line-level data, without any post-processing.