applied sciences

MDPI

Article

OCR4all—An Open-Source Tool Providing a (Semi-)Automatic OCR Workflow for Historical Printings

Christian Reul [1,*], Dennis Christ [1], Alexander Hartelt [1], Nico Balbach [1], Maximilian Wehner [1], Uwe Springmann [2], Christoph Wick [1], Christine Grundig [3], Andreas Büttner [4] and Frank Puppe [1]

1 Artificial Intelligence and Applied Computer Science, University of Würzburg, 97074 Würzburg, Germany; dennis.christ@stud-mail.uni-wuerzburg.de (D.C.); alexander.hartelt@uni-wuerzburg.de (A.H.); nico.balbach@uni-wuerzburg.de (N.B.); maximilian.wehner@uni-wuerzburg.de (M.W.); christoph.wick@uni-wuerzburg.de (C.W.); frank.puppe@uni-wuerzburg.de (F.P.)
2 Center for Information and Language Processing, LMU Munich, 80538 Munich, Germany; uwe@springmann.net
3 Institute for Modern Art History, University of Zurich, 8006 Zurich, Switzerland; christine.grundig@uzh.ch
4 Institute for Philosophy, University of Würzburg, 97074 Würzburg, Germany; andreas.buettner@uni-wuerzburg.de
* Correspondence: christian.reul@uni-wuerzburg.de

Received: 6 September 2019; Accepted: 1 November 2019; Published: 13 November 2019

Abstract: Optical Character Recognition (OCR) on historical printings is a challenging task mainly due to the complexity of the layout and the highly variant typography. Nevertheless, in the last few years, great progress has been made in the area of historical OCR, resulting in several powerful open-source tools for preprocessing, layout analysis and segmentation, character recognition, and post-processing. The drawback of these tools often is their limited applicability by non-technical users like humanist scholars and in particular the combined use of several tools in a workflow. In this paper, we present an open-source OCR software called OCR4all, which combines state-of-the-art OCR components and continuous model training into a comprehensive workflow. While a variety of materials can already be processed fully automatically, books with more complex layouts require manual intervention by the users. This is mostly due to the fact that the required ground truth for training stronger mixed models (for segmentation, as well as text recognition) is not available, yet, neither in the desired quantity nor quality. To deal with this issue in the short run, OCR4all offers a comfortable GUI that allows error corrections not only in the final output, but already in early stages to minimize error propagations. In the long run, this constant manual correction produces large quantities of valuable, high quality training material, which can be used to improve fully automatic approaches. Further on, extensive configuration capabilities are provided to set the degree of automation of the workflow and to make adaptations to the carefully selected default parameters for specific printings, if necessary. During experiments, the fully automated application on 19th Century novels showed that OCR4all can considerably outperform the commercial state-of-the-art tool ABBYY Finereader on moderate layouts if suitably pretrained mixed OCR models are available. Furthermore, on very complex early printed books, even users with minimal or no experience were able to capture the text with manageable effort and great quality, achieving excellent Character Error Rates (CERs) below 0.5%. The architecture of OCR4all allows the easy integration (or substitution) of newly developed tools for its main components by standardized interfaces like PageXML, thus aiming at continual higher automation for historical printings.

Keywords: optical character recognition; document analysis; historical printings

1. Introduction

While Optical Character Recognition (OCR) is regularly considered to be a solved problem [1], gathering the textual content of historical printings using OCR can still be a very challenging and cumbersome task [2], due to various reasons. Among the problems that need to be addressed for early printings is the often intricate layout containing images, ornaments, marginal notes, and swash capitals. Furthermore, the non-standardized typography represents a big challenge for OCR approaches. While modern fonts can be recognized with excellent accuracy by so-called omnifont or polyfont models, very early printings like incunabula (books printed before 1501), but also handwritten texts usually require book-specific training in order to reach Character Error Rates (CERs) well below 10% or even 5%, as shown by Springmann et al. [3] (printings) and Fischer et al. [4] (manuscripts). For a successful supervised training process, the Ground Truth (GT) in the form of line images and their corresponding transcriptions has to be manually prepared as training examples.

In the last few years, some progress has been made in the area of historical OCR, especially concerning the character recognition problem. An important milestone was the introduction of recurrent neural networks with Long Short Term Memory (LSTM) [5] trained using a Connectionist Temporal Classification (CTC) [6] decoder, which Breuel et al. applied to the task of OCR [7]. The LSTM approach was later extended by deep Convolutional Neural Networks (CNN), pushing the recognition accuracy even further [8,9].

The present paper describes our efforts to collect these recent advances into an easy-to-use and platform independent software environment called OCR4all that enables an interested party to obtain a textual digital representation of the contents of these printings. OCR4all covers all steps of an OCR workflow from preprocessing, document analysis (segmentation of text and non-text regions on a page), model training, to character recognition of the text regions. Our focus is throughout on an easy-to-use and efficient method, employing automatic methods where feasible and resorting to manual intervention where necessary. A special feature of our process model is that manual interventions lead to the production of high quality GT being used as additional training data, thus enabling a spiral towards continuously higher automation. In the following, we give a short overview over the steps of a typical OCR workflow and how we address the challenges that arise for early printings.

1.1. Steps of a Typical OCR Workflow

The character recognition in itself only represents one subtask within an OCR workflow, which usually consists of four main steps (see Figure 1), which often can be split up into further substeps. We use the term "OCR" as a separate main step within the OCR workflow, as other notations like "recognition" would be misleading since the step comprises more sub-tasks than the text recognition alone.

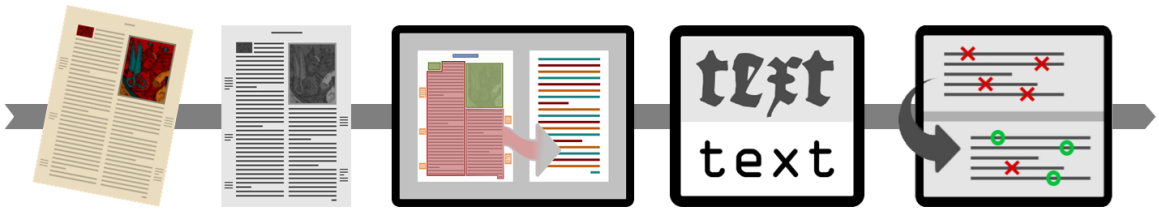

Figure 1. Main steps of a typical OCR workflow. From left to right: original image, preprocessing, segmentation, OCR, postcorrection.

1. Preprocessing: First of all, the input images have to be prepared for further processing. Generally, this includes a step that simplifies the representation of the original color image by converting it into binary, as well as a deskewing operation in order to get the pages into an upright position. Additional routines like cropping, dewarping, denoising, or despeckling may be performed.

2. Segmentation: Next, one or several segmentation steps have to be conducted, mostly depending on the material at hand and the requirements of the user. After separating the text regions from non-text areas, individual text lines or even single glyphs have to be identified. Optionally, a more fine-grained classification for non-text (images, ornaments, etc.), as well as for text elements (headings, marginalia, etc.) can be performed already on the layout level. Another important sub-task is the determination of the reading order, which defines the succession of text elements (region and/or lines) on a page.
3. OCR: The recognition of the segmented lines (or glyphs) leads to a textual representation of the printed input. Depending on the material at hand and the user requirements, this can either be performed by making use of existing mixed models and/or by training book-specific models after producing the required GT.
4. Post-processing: The raw OCR output can be further improved during a post-processing step, for example by incorporating dictionaries or language models. This step can support or replace the manual final correction phase depending on the accuracy requirements of the users.

As for the final output, plain text, that is the (post-processed) OCR output, has to be considered the minimal solution. Additionally, several formats that can incorporate a more sophisticated output also containing layout or confidence information have been proposed, for example ALTO (https://www.loc.gov/standards/alto), hOCR [10], or PAGE [11].

1.2. Challenges for the Users

To produce training data for the OCR, one has to find and transcribe text lines (considering a line-based approach) manually, which is a highly non-trivial task when dealing with very old fonts and historical languages. However, the combination of all steps for automatic transcription with manual support can be supported by components of open-source tools such as OCRopus, Tesseract, or Calamari. While these tools are highly functional and very powerful, their usage can be quite complicated, as they:

- in most cases lack a comfortable GUI, which leaves the users with the often unfamiliar command line usage
- usually rely on different input/output formats, which requires the users to invest additional effort in order to put together an end-to-end OCR workflow
- sometimes require complicated and error prone installation and configuration procedures where, e.g., the users have to deal with missing dependencies
- have a steep learning curve (at least for non-technical users)

These aspects are particularly problematic for inexperienced users with limited technical background like humanities scholars to produce machine actionable text and GT from scans of historical printings. Therefore, providing adequate interfaces with an entire workflow available for automatic transcription, as well as for manual correction by non-technical users is of key importance, since most tools usually do not cover the entire workflow described above, at least not in a satisfactory manner, but rather excel on smaller sub-tasks.

1.3. OCR4all

To deal with these issues, we present our open-source tool OCR4all (https://www.uni-wuerzburg.de/en/zpd/ocr4all), which aims to encapsulate a comprehensive OCR workflow into a single Docker [12] application, ensuring easy installation and platform independency. The goal is to make the capabilities of state-of-the-art tools like OCRopus or Calamari available within a comprehensible and applicable semi-automatic workflow to basically any given user with the option to decide on different compromises between the resulting accuracy and manual effort. This is achieved by supplying the users with a comfortable and easy-to-use GUI and a modular approach, allowing for an efficient

correction process in between the various workflow steps in order to minimize the negative effects of consequential errors. Another important aspect is the option to reduce iteratively the CER by constantly retraining the recognition models on additional training data, which have been created during the processing of a given book. The user can choose to invest a substantial amount of manual effort, e.g., for a high quality edition, or little or no effort for a largely automatic transcription for a subsequent quantitative analysis of the text, in particular for later and more uniform prints.

In addition to the code, a setup guide and a comprehensive step-by-step user manual, together with some example data are available on GitHub (https://github.com/OCR4all/getting_started). Moreover, there is a mailing list (https://lists.uni-wuerzburg.de/mailman/listinfo/ocr4all) where we inform about the latest developments and new version releases.

1.4. Outline of This Work

The paper is structured as follows: First, Section 2 provides an overview of important contributions concerning OCR relevant to our task. In Section 3, we thoroughly describe OCR4all including the overall workflow and the single submodules. Next, we perform several experiments on a variety of historical printings. The results are discussed in Section 5 before Section 6 concludes the paper by summing up the insights and pointing out our goals for the future of OCR4all.

2. Related Work

In this section, we give a brief overview of OCR related tools and topics. First, we focus exclusively on the OCR since it has to be considered the core task of the entire workflow. Afterwards, we discuss tools that (aim to) provide an entire OCR workflow.

2.1. Optical Character Recognition

We chose Calamari [13] as our OCR engine since it is available under an open-source license and previous tests have demonstrated its advantages compared to other OCR engines regarding recognition capabilities and speed [9]. It focuses solely on the OCR training and recognition step using Tensorflow [14] as its backend and does not offer any preprocessing, segmentation, or post-processing capabilities, which is why it will not be covered in the upcoming comprehensive discussion of tools that provide a full OCR workflow.

Evaluation of OCR Engines

A thorough comparison of a shallow LSTM (OCRopus 1) and a deep CNN/LSTM hybrid (Calamari) was given in [9]. Three early printed books, printed between 1476 and 1505 in German and Latin, were used as training and evaluation data. The results showed, as anticipated, that the advantage of the deep network grew with an increasing number of lines used for training, yielding an average improvement in CER of 29% for 60 lines and 43% for 1000 lines.

In [13], Wick et al. compared Calamari, OCRopus 1, OCRopus 3, and Tesseract 4 on two public datasets: first, the UW3[15] and the DTA19dataset, which is a part of the GT4HistOCRcorpus (https://zenodo.org/record/1344132). On the UW3 dataset, Calamari achieved a CER of 0.155%, considerably outperforming OCRopus 1 (0.870%), OCRopus 3 (0.436%), and Tesseract 4 (0.397%). The inclusion of confidence voting improved Calamari's result by another 26% to a CER of just 0.114%. Evaluations on the DTA19 dataset led to similar observations, with Calamari reaching a CER of 0.221% (0.184% with voting) compared to the significantly higher 1.59% of OCRopus 1 and 0.907% of OCRopus 3. When using a GPU, Calamari required 8 ms to train and 3 ms to predict a line, which proved to be considerably faster than OCRopus 3 (10 ms and 7 ms), while OCRopus 1 (850 ms and 330 ms) and Tesseract 4 (1200 ms and 550 ms) were far behind due to their lack of GPU support.

A case study [16] dealing with German 19th Century Fraktur scripts from various source materials (mostly novels, but also a journal, different volumes of a newspaper, as well as a dictionary) was performed. Mixed models for Calamari and OCRopus 1 were produced by training on a very extensive

corpus of German Fraktur data from the 19th Century, which was completely distinct from the material of the evaluation set. Altogether, ABBYY, despite making full use of its post-correction capabilities, achieved a CER of 2.80%, but was significantly outperformed by the raw OCR output of Calamari (0.61%) and even OCRopus 1 (1.90%).

Conclusion

Based on the results presented above and our personal experience, Table 1 sums up and rates the capabilities of the most important available OCR engines in terms of historical OCR.

Table 1. Comparison and rating of the capabilities of four modern OCR engines.

Step	ABBYY	OCRopus 3	Tesseract 4	Calamari
Recognition	✓✓	✓✓	✓✓	✓✓✓
Training	✓	✓✓	✓	✓✓✓
Manual Correction	✓✓✓	✗	✗	✗

As for the recognition, the main criteria are accuracy and speed. Since we consider post-processing using dictionaries and language models to be an individual step in the workflow, we rate the raw recognition capabilities of the engines. Due to the results presented above, the best rating goes to Calamari.

Regarding the training, we rate the engines mainly based on their speed and effectiveness, but also take into account the user friendliness when it comes to training on real data. OCRopus 3, Tesseract 4, and Calamari in general allow pairs of line images and their transcriptions as training input, which is very comfortable and straightforward for the user. While Calamari can deal with the image/text pairs directly, just like OCRopus 1, OCRopus 3 requires creating a .tar file comprising the data. As for Tesseract 4, the training of models on real historical data has been considered at least impracticable for several years until recently, a solution was discovered and made publicly available (https://github.com/OCR-D/ocrd-train). However, this requires an extension to the standard training tools. While it is basically possible to train single glyphs and consequently a book-specific model using ABBYY, this is a tedious and ineffective task that seems to be mainly geared towards the recognition of quite specific ornament letters. This effectively limits the recognition capability to the expensive existing historical models one has to license from ABBYY.

ABBYY offers a comprehensive set of support tools for the manual postcorrection including a synoptic image/text view, markers for possible errors based on recognition confidence and dictionaries, and a selection of possible alternatives. While OCRopus 1 at least allows creating a browser based synoptic view, OCRopus 3, Calamari, and Tesseract 4 do not offer any form of user interaction regarding the correction of the OCR output.

2.2. Tools Providing an OCR Workflow

Before discussing the various OCR workflow tools in detail, we first give an overview of their respective capabilities in Table 2. Regarding the OCR step, we mostly incorporated the ratings from Table 1 since, as shown above, there are several detailed comparative evaluations available that the other steps are lacking. As Calamari represents our main OCR engine, we adopted its ratings. There is a single exception: since OCR4all offers a line-based synoptic correction view including some user conveniences like a customizable virtual keyboard, but currently lacking a dictionary or confidence based error detector, we adjusted the rating for manual correction accordingly.

In the following, we discuss the four tools from Table 2.

Table 2. Comparison of existing tools providing an OCR workflow with OCR4all.

Step	Sub Task	ABBYY	OCRopus 3	Tesseract 4	OCR4all
Preprocessing	Deskewing	✓	✓	✓	✓
	Binarization	✓	✓	✓	✓
Segmentation	Image/Text	✓	✗	✓	✓
	Semantic Distinction	✗	✗	✗	✓
	Line Segmentation	✓	✓	✓	✓
	Reading Order	✓	✓	✓	✓
	Manual Correction	✓	✗	✗	✓
Historical OCR	Recognition	✓✓	✓✓	✓✓	✓✓✓
	Training	✓	✓✓	✓	✓✓✓
	Manual Correction	✓✓✓	✗	✗	✓✓
Post-processing	Dictionaries	✓	✗	✓	✗
	Language Modeling	✓	✗	✓	✗
open-source	-	✗	✓	✓	✓

2.2.1. ABBYY

At least on contemporary material, the proprietary ABBYY OCR engine (https://www.abbyy.com) clearly defines the state-of-the-art for preprocessing, layout analysis, and OCR. A wide variety of documents with considerably differing layouts can be processed by the fully automated segmentation functionality, whose results can be manually corrected, if necessary.

Especially regarding the character recognition, ABBYY's focus clearly lies on modern printings since this represents their bulk business. Currently (July 2019), their products support close to 200 recognition languages offering strong language models and dictionary assistance for about a quarter of them. Despite the focus on modern prints, the repertoire also includes the recognition of historical European documents and books printed in six languages.

Apart from its closed source and proprietary nature, ABBYY's shortcomings in the area of OCR of (very) early printings lead to the conclusion that it does not fit the bill despite its comprehensive and powerful preprocessing, segmentation, and recognition capabilities (on later material), as well as its easy setup and comfortable GUI.

2.2.2. Tesseract

Just like ABBYY, the open-source OCR engine Tesseract [17] provides a full OCR workflow including built-in routines for preprocessing like deskewing and binarization, as well as for layout analysis, but overall, it is significantly less successful than ABBYY. Tesseract's OCR training and recognition capability recently (Version 4.0+) have improved considerably due to the addition of a new OCR engine based on LSTM neural networks. A wide variety of mixed models for different languages and scripts are openly available on the project's GitHub repository. Similar to ABBYY and contrary to OCRopus 1/2/3 and Calamari, Tesseract supports the use of dictionaries and language modeling. While Tesseract has its strengths in the fully automatic out-of-the-box processing of modern texts, it falls short when it comes to historical material.

2.2.3. OCRopus

The open-source toolbox OCRopus 1 (https://github.com/tmbdev/ocropy) [18] comprises several Python-based tools for document analysis and recognition. This includes highly performant algorithms for deskewing and binarization, as well as a segmentation module that extracts text lines from a page in reading order. While the segmentation can quite comfortably deal with modern standard layouts, it tends to struggle with typical historical layouts with marginalia, swash capitals, etc. When a page has already been split up into regions, however, the line segmentation usually identifies the single lines very reliably and accurately, at least when working with Latin script. OCRopus 1 was the first

OCR engine to implement the pioneering line based approach for character recognition introduced by Breuel et al. [7] using bidirectional LSTM networks. Furthermore, this method significantly simplified the process of training new models since the user just has to provide image/text pairs on the line level, which can be created by using an HTML-based transcription interface in a browser. Despite the outdated shallow network structure, OCRopus 1 still proves to be a cornerstone for OCR workflows dealing with historical printings mainly for two reasons. First, the preprocessing usually achieves excellent results due to its robust deskewing approach, as well as its adaptive thresholding technique used for preprocessing [19]. The second is due to the robust line segmentation described above.

After the comparatively disregarded OCRopus 2 (https://github.com/tmbdev/ocropy2), the third edition of OCRopus 3 (https://github.com/NVlabs/ocropus3) was released in May 2018. It introduced a PyTorch backend [20], which enabled the utilization of deep network structures and GPU support, resulting in better recognition rates and faster training and prediction. Concerning the other steps in the OCR workflow like binarization, deskewing, and segmentation, OCRopus 3 almost exclusively relies on deep learning techniques. To the best of our knowledge, there are not yet any comparisons available between the traditional methods of OCRopus 1 and the new approach by OCRopus 3.

3. Methods

In this section, we focus on the OCR4all software. After introducing the data structure, we first describe the workflow and its individual modules, including their input/output relations, in detail, before we look at the encapsulating web GUI, which offers various possibilities to influence the workflow by manual corrections or configurations.

3.1. Software Design and Data Structure

We chose to implement the workflow as a server application accessible by a web app because this allows a deployment as a true web app, as well as locally. Furthermore, the incorporation of Docker effectively assures platform independency, as it can be installed and run on basically all modern operating systems including Windows, Mac, and Linux.

Regarding our data structure, apart from different representations of page images, we focus on PageXML [11] as the main carrier of information. This allows for a modular integration of the main submodules of OCR4all and sets up easy to fulfill requirements regarding interfaces, ensuring a reasonably straightforward addition of new submodules or replacement of existing ones. Additionally, defining a unified interface for all tools and modules enables the usage of a comprehensive post-processing functionality. Another positive side effect of this approach is that submodules developed by us for OCR4all can be integrated analogously into other OCR workflows that use PageXML.

PageXML requires one XML file per page, which can store a wide variety of information, most importantly:

- A page can comprise an arbitrary number of regions whose reading order can be specified.
- Among others, a region can store its enclosing polygon and type.
- There are main types like image, text, or music. Text regions can be further classified into sub-types like running text, heading, page number, marginalia, etc.
- A region can contain an arbitrary number of text lines.
- Each line stores its enclosing polygon, as well as an arbitrary number of text elements, which may contain GT, various OCR outputs, normalized texts, etc.

3.2. OCR4all Workflow

Figure 2 shows the steps of the workflow implemented in OCR4all. After acquiring the scans and an optional preparation step, for example by using ScanTailor (https://scantailor.org/), the original images can be placed into the workspace. Next, image preprocessing is applied to the scans before

several steps, like region segmentation and extraction, as well as line segmentation, producing line images required as input for character recognition or ground truth production. The output of character recognition can either directly serve as the final result or can be corrected by the user, which enables the training of more accurate book-specific models, yielding better recognition results.

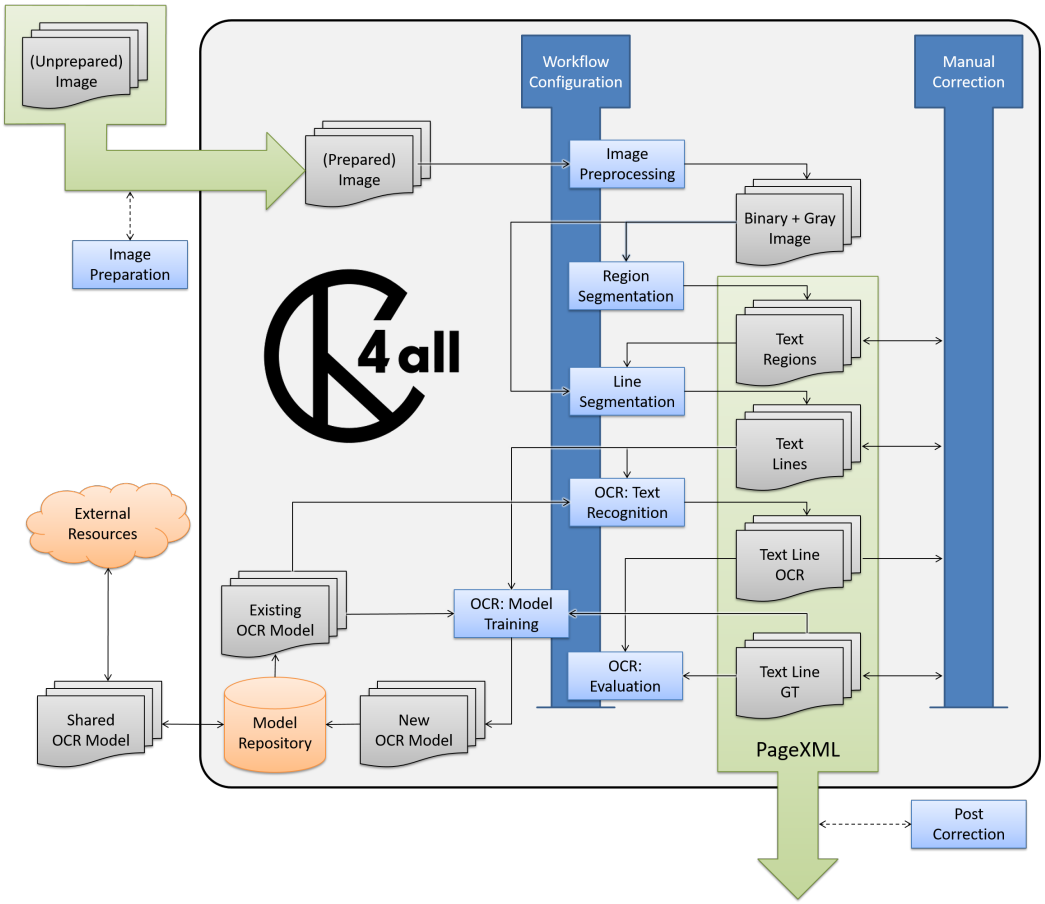

Figure 2. The main steps of the OCR4all workflow, as well as the optional image preparation and post-correction steps that are not part of the main tool (yet).

In the following, we discuss a typical workflow by going through the four main steps from Figure 1 and discuss the corresponding modules in OCR4all as shown in Figure 2. Furthermore, we always state the input and output relation of each module by describing the actual data each module works on, which is often produced by combining the information stored as PageXML with the preprocessed grayscale or binary image.

3.3. Preprocessing

In the preprocessing main step, the input images are prepared for further processing. Before the two standard sub-tasks binarization and deskewing take place, an optional external preparation step can be performed.

Image Preparation

Input: unprepared image (image containing two scanned pages, a page rotated in an invalid orientation, etc.)
Output: prepared image (single page in an upright position)

OCR4all expects the input images to be in an upright position and already segmented into single pages, which can easily be achieved by using ScanTailor. Furthermore, it is recommended to remove excessive amounts of scan background, although this is not mandatory. Figure 3 shows an example of a valid and an invalid input, which also represents a possible input and output of ScanTailor. In fact, ScanTailor is not a true OCR4all submodule, since it cannot be integrated due to the lack of a web based user interface. However, we still decided to list it as a module since this step belongs to the workflow and the input images have to be added from external sources anyway. It is possible to deal with unprepared images like the ones described in the input completely within OCR4all, but it certainly is not the recommended course of action.

Figure 3. An example input and output of the image preparation and preprocessing steps. The image on the left represents an undesirable input for OCR4all, while the ScanTailor output in the middle is completely sufficient. During the preprocessing step, the skewed color image in the middle is transformed into the deskewed binary image on the right.

During the preprocessing substep, the input image gets converted into a binary and (optionally) a normalized grayscale image. Additionally, a deskewing operation can be performed. See Figure 3 for an example input and output of this step. For both steps, we used the methods implemented in the OCRopus 1 *nlbin* script.

3.4. Segmentation

During the segmentation main step, the preprocessed images are first segmented into regions. Then, after extracting the ones containing text, the line segmentation is performed.

3.4.1. Region Segmentation

Input: preprocessed image
Output: structural information about regions (position and type) and their reading order

The general goal of this step is to identify and optionally classify regions in the scan. There are different manifestations that considerably impact the complexity of the task and entirely depend on the material at hand, the use case, and the individual requirements of the user. For example, when the goal is to gather and process a book for editorial purposes, it is mandatory to obtain a flawless text, and consequently, a (close to) flawless segmentation result is advisable. Additionally, in these cases, it is often desired to perform a semantic classification of text regions already on the layout level. Therefore, a considerable amount of human effort has to be expended. On the contrary, the most simplistic approach would be to only distinguish between text and non-text regions in order to obtain a good OCR result. For both scenarios, OCR4all offers viable solutions, which will be briefly explained in the following.

For moderate layouts, OCR4all offers the so-called Fully Automatic Segmentation (FAS), which focuses on the identification of non-text elements and, apart from that, considers the entire page

as a single running text segment (see Figure 4, right). The first step is similar to the one used in LAREX: After identifying the Connected Components (CC), the dimension of a typical letter are calculated. Then, the algorithm looks for unusually large CCs to identify images, partly or as a whole. For historical works, where images are surrounded by a frame in the vast majority of cases, this rather straightforward approach works very well, especially when incorporating several heuristics, like combining overlapping image rectangles into new ones in a bottom-up approach. Detected images, borders, swash capitals, etc., are marked in PageXML.

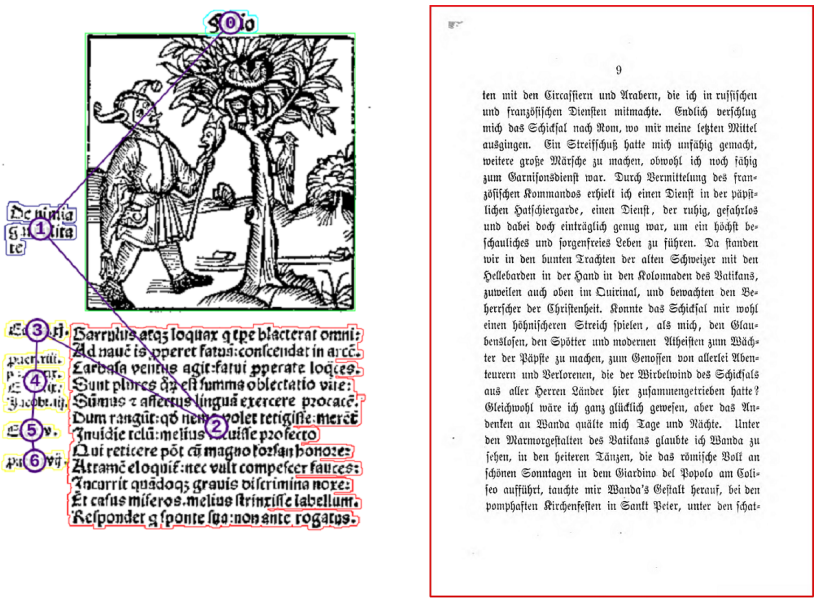

Figure 4. Left: a LAREXsegmentation output consisting of an image (green), running text (red), marginalia (yellow), image caption (blue), and the page number or folio identifier (cyan), as well as the reading order. Right: output of the FSAfor a standard 19th Century novel layout.

While admittedly, this is a quite simplified approach, our experiences using the FAS have been very positive for several reasons. For example, there is a rudimentary implicit text/non-text segmentation available, as well as a highly performant column detection functionality. Furthermore, because of the aforementioned capabilities of the line segmentation, the FAS can actually be applied to an unexpectedly wide variety of historical printings, runs fully automatically, and basically at very low cost. On the downside, this approach does not perform any kind of meaningful semantic distinction of text parts.

Naturally, we have been experimenting with more sophisticated approaches like pixel classifiers, but the lack of fitting GT, both in terms of quantity, but also quality, led to unsatisfying results, at least when targeting a mixed model as a generic solution. This topic will be addressed in greater detail during the Discussion and the Future Work Section.

A way to deal with complex layouts, especially when a fine grained semantic distinction is desired (see Figure 4, left), the in-house development tool LAREX (https://github.com/OCR4all/LAREX) represents the means of choice. It offers the user a variety of automatic, assisted, and fully manual tools that allow gathering a complex page layout with reasonable effort. It is worth mentioning that it is also possible to load existing segmentation results into LAREX and mostly use it as an editor by comfortably correcting the results, if necessary. The drawback of LAREX is that, at least as of now, it expects the user to have a look at every page and approve each result individually. Of course, while checking each page is almost imperative for very complex layouts and high user expectations, it cannot be considered the preferred solution for considerably easier or even trivial layouts.

Naturally, the user can decide on a page-by-page basis which segmentation approach to apply. For example, if a book starts with a quite complicated title page and a complex register, both in terms

of layout, but apart from it consisting of pages with a trivial one-column layout, the user can easily segment the first few pages using LAREX and then switch to the FSA for the remainder of the book. Due to the well defined interfaces, all preceding and subsequent steps can be applied to all pages in the exact same manner and without any further differentiations.

3.4.2. Line Segmentation

Input: text region regions
Output: extracted text lines

The actual line segmentation operates on individual region images, if available, instead of the entire page. Therefore, the text regions identified during the segmentation step need to be extracted from the page images, which is done by a region extraction substep. We cut out the polygons stored in the PageXML file from the corresponding binary image. After the extraction, the region images are separately deskewed by applying the OCRopus 1 nlbin script. Processing the regions one-by-one can lead to considerably better results than the standard deskewing on page level since areas/regions can be skewed independently of each other. The line segmentation is performed by applying an adapted version of the Kraken (https://github.com/mittagessen/kraken) line segmentation script to each extracted region individually.

The output produced by the applied algorithm is considerably more complex than just the bounding rectangle of a text line: After assigning (parts of) CCs to their respective text lines, their parts are connected to a tight-fitting polygon in order to produce an optimal line segmentation result. To achieve this, we extended the Nashi (https://github.com/andbue/nashi) line segmentation wrapper. An extreme example of the segmentation capabilities is shown at the bottom of Figure 5.

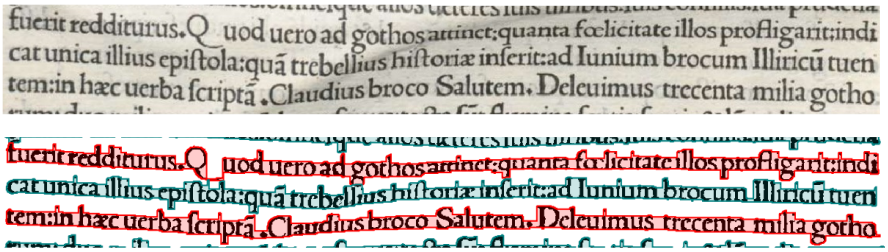

Figure 5. Example of the line segmentation step. Top: input image of a very challenging text snippet. Bottom: high quality line segmentation result showing the individual line polygons. For reasons of clarity, altering lines are depicted in different colors.

3.5. OCR

After obtaining the text lines, the OCR main step can be performed including the character recognition either by applying available mixed models or the results of a book-specific training. Furthermore, an error analysis of produced outputs is provided, which just like the training, requires GT that can be produced by manual correction, which we will discuss later on.

3.5.1. Character Recognition

Input: text line images and one or several OCR models
Output: textual OCR output on the line level

After segmenting the pages into lines, it is now possible to perform OCR on the results. As of now, Calamari is the only OCR engine that is integrated into OCR4all by default. However, due to the well defined interfaces, additional engines can be added and operated with manageable effort.

OCR4all comes with four single standard models (https://github.com/Calamari-OCR/ocr4all_models). Since voting ensembles have proven to be very effective, we additionally provide a full set of

model ensembles (https://github.com/Calamari-OCR/calamari_models) consisting of five models for each. Calamari supports the utilization of an arbitrary number of models, and the confidence voting is automatically triggered when more than one model is provided.

3.5.2. Model Training

Input: line images with corresponding GT, optionally already existing models to build upon
Output: one or several OCR models

The model training that allows training book-specific Calamari models is not only one of the most central modules of the entire workflow, but also probably the most complex and challenging one when it comes to enabling non-technical users to utilize all available features. The training algorithm possesses a variety of hyper-parameters, which have either been set to a fitting default value or are derived from the amount of training data.

Our main goal for the training module was to provide a non-technical user with the ability to make use of all the available accuracy improving techniques comfortably like cross-fold training to produce voting ensembles, pretraining to use existing models as a starting point, and data augmentation to generate additional training examples.

Iterative Training Approach

To keep the manual effort to a minimum, we introduced an Iterative Training Approach (ITA), which is fully supported by OCR4all. The general idea is to minimize the required human workload by increasing the computational load. Correcting an existing OCR output with a (very) good recognition accuracy is (considerably) faster than transcribing from scratch or correcting a more erroneous result. Consequently, we aim to get to a reasonable recognition accuracy quickly, which allows for an efficient GT production process. Therefore, we integrated an ITA whose procedure is listed in the following:

1. Transcribe a small number of lines from scratch or correct the output of a suitable mixed model, if available.
2. Train a book specific model/voting ensemble using all available GT that have been transcribed up to this point, including earlier iterations.
3. Apply the model/voting ensemble to further lines.
4. Correct the output.
5. Repeat Steps 2–4 until the desired recognition accuracy is reached or the entire book is transcribed. An approximation of the current accuracy is given by the error analysis (see the next section).

Since the ITA, especially when combined with cross-fold training, can quickly produce plenty of OCR models, a certain amount of bookkeeping is required to stay on top of things. Therefore, OCR4all provides an intuitive automatic naming convention for the trained models.

3.5.3. Error Analysis

Input: line-based OCR predictions and the corresponding GT
Output: CER and confusion statistics

To enable an objective assessment of the recognition quality achieved by the models at hand, we incorporated the Calamari evaluation script into OCR4all. For a given selection of pages, it compares the OCR results to the corresponding GT and calculates the CER using the Levenshtein distance. Additionally, a confusion table displaying the most common OCR errors and their frequency of occurrence is provided.

3.6. Result Generation

Input: GT and OCR results
Output: final output as text files

For the average OCR4all user, PageXML most likely does not represent the desired output format that is needed for further processing with other tools. Consequently, we also offer a simple textual output where the line-based OCR results (or the GT) are concatenated in reading order and stored as a text file. To preserve additional information like semantic classes, it is of course also possible to keep the PageXML files.

3.7. Manual Corrections

Input: images and their corresponding PageXML files
Output: corrected PageXML files

As emphasized during the Introduction, a fully automated workflow is often not reasonable or at least cannot be expected to yield sufficient (depending on the use case) or even perfect results, especially when dealing with early printings. Consequently, a potent, flexible, comprehensible, and easy-to-use option for manual correction is a must-have for every OCR workflow tool that relies on user intervention. In OCR4all, this core task is covered by LAREX, whose functionality has been considerably extended since its original release as a region segmentation tool and will be explained in the following.

LAREX works directly on PageXML files and the corresponding images. After loading a page, the information is displayed using additional layers over the image in three different views, which are all interconnected with each other:

- Regions: LAREX offers a wide variety of tools and procedures to create new and edit existing regions, including the adding and deletion of regions, as well as changing their (sub-)type and sophisticated polygon manipulation operations.
- Lines: From an editing point of view, lines are treated exactly like regions and therefore allow the same comprehensive set of operations with minor adaptions.
- Text: The text view (see Figure 6) is divided into two further sub-views. In the first one, the page image is still presented to the user with all text lines color-coded, indicating the availability of corresponding GT. While this view is a suitable solution for users that aim for a perfect text and consequently have to take a thorough look at each line anyway, it is not optimal for use cases where users just want to scan the pages and lines quickly for obvious mistakes. For this use case, we introduced a second sub-view, which optimizes the correction process by providing a synoptic view where an editable text field is placed directly under each line image, allowing the user to get a quick overview.

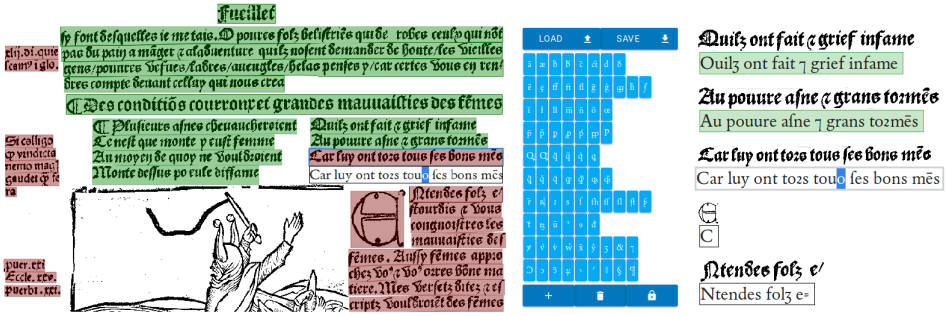

Figure 6. Two text correction views in LAREX: page based view with a virtual keyboard where a selection of lines can be corrected (left and center) and the corresponding line based view (right).

Configurations

To be able to deal with the wide variety of printings and the distinct challenges proposed by them, as well as to satisfy the individual needs of each user, OCR4all offers plenty of ways to influence not only the workflow in itself, but also the parameters of the single submodules. All configurations are entirely accessible from the web GUI in an intuitive way and do not require any kind of knowledge regarding the usage of the command line.

In order to not overwhelm inexperienced users by confronting them with a plethora of confusing options, settings, and parameters, but also to provide more experienced users to adapt selected settings to optimize the results, we carefully split the available options for each submodule into general and advanced settings. While the general settings usually only contain one or two parameters whose default settings normally completely suffice for the average user, the advanced settings comprise all remaining parameters and allow experienced users to maintain full control.

In particular, the degree of automation can be influenced via the so-called process flow. By default, all modules from preprocessing to recognition are configured and executed at once. However, if, for example, a segmentation using LAREX is needed, which requires user intervention, the subsequent steps can still be run at once and fully automatically.

4. Evaluations

To evaluate the effectiveness and usability of OCR4all, we performed several experiments on various books using different evaluation settings, which we will discuss in the following. After introducing the data and evaluating the fully automated processing of newer works, we focus on the precise text recognition of early printed books. Then, we take a closer look at the effects of the ITA. Afterwards, we experiment with a reduced degree of manual intervention by the user by evaluating a less costly, but also less precise segmentation approach.

The main goals of our experiments are to evaluate the

- performance of OCR4all when applied in a fully automated way.
- manual effort required to capture a book using a precise segmentation and aiming for a very low error rate (<1% CER) dependent on the complexity of the material and the experience of the user.
- speedup when incorporating the ITA.
- potential speedup when considerably lowering the requirements regarding segmentation, especially considering the fine-grained semantic distinction of layout elements.

4.1. Data

In this section, we briefly introduce the books we used for our experiments comprising 19th Century Fraktur novels and a variety of early printed books.

4.1.1. 19th Century German Novels Printed in Fraktur

The first part of our evaluation corpus consists of 19th Century German novels (with one exception from the late 18th Century), which are currently collected and OCRed by the Chair for Literary Computing and German Literary History of the University of Würzburg. The overall quality of the material varies considerably as shown in Figure 7. The less complex layout, the more regular typography, and the desired use for quantitative experiments make it neither necessary, nor feasible to invest an extensive amount of manual work. A highly automated workflow is intended instead.

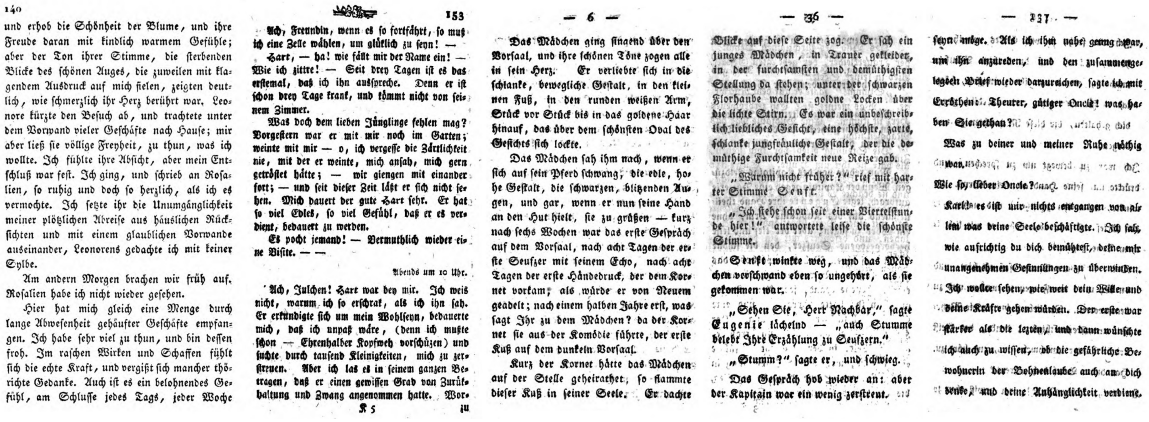

Figure 7. Example images of the German novel corpus. From left to right: F1870, F1781, F1818 (page in decent condition), F1818 (page in bad condition), F1803.

4.1.2. Early Printed Books

The second part of our evaluation corpus consists of books printed before 1600 including five editions of the *Narrenschiff* (cf. the Narragonien digital project: http://kallimachos.de/kallimachos/index.php/Narragonien; used languages: Latin, German, French, and Dutch), 17 works related to the influential early modern universal scholar Joachim Camerarius the Elder (cf. the Opera Camerarii project: http://wp.camerarius.de; used languages: Latin and Greek), where we focused on the OCR of the Latin parts and just marked Greek text (embedded parts of Greek, mostly scientific technical terms) with a placeholder for later processing, and three further German early modern printings worked on during a practical course. In the following, we refer to the individual books with a shortcut (N, C, or P) combined with the year of publication, for example C1566 for a work from Camerarius printed in 1566. Figure 8 shows representative example images of some of the books used, as well as some desired segmentations. For reasons of clarity, we refrained from depicting the reading order.

Figure 8. Example images of early printed books we used for evaluations. Top (from left to right): four Camerarius works C1566, C1541, C1563, and C1563 segmented; middle left: two works from the practical course: P1484 and P1509; and six editions of the Narrenschiff, right: N1506 and N1506 segmented and bottom: N1549, N1494, N1499, and N1499 segmented.

4.2. Fully Automated Processing of 19th Century Novels

First, we reduced the manual effort to a minimum by choosing a fully automated approach. Since the FAS of OCR4all can comfortably deal with moderate layout, we first focus our in-detail evaluation on the 19th Century Fraktur novels, before we turn to more complex material.

This experiment was performed on ten German Fraktur novels from the corpus described above using the Calamari Fraktur 19th Century ensemble, which was trained on a wide variety of data also derived from 19th Century Fraktur novels (see [16] for details).

We randomly selected ten pages from each novel and processed them fully automatically with OCR4all, as well as with ABBYY Finereader Engine CLI for Linux (https://www.ocr4linux.com/en) Version 11 together with ABBYY's historical Fraktur (Gothic) module and Old German language settings. The results were compared by calculating the CER on a page-to-page basis. To ensure a fair

comparison, several regularizations, for example the normalization of the long and short version of the *s*, were performed beforehand.

4.2.1. Results

Table 3 sums up the results. The values show that OCR4all considerably outperformed ABBYY Finereader on every single book resulting in an average improvement of over 84% and a relative improvement of almost eight with respect to the error rate. On eight of the ten books, CERs of below 1% were achieved, while six books even yielded error rates below 0.5%. Wild fluctuations in CER can be observed for ABBYY Finereader (best: 0.48%, worst: 27%), but also for OCR4all (best: 0.06%, worst: 4.89%) caused by the highly variant quality of the scans, as shown in Figure 7.

Table 3. CERs achieved by ABBYY Finereader and OCR4all when being applied fully automatically to different books. The final two columns indicate the percent error reduction *ErrRed.* and the improvement factor *Impr.* yielded by OCR4all over ABBYY. Furthermore, we provide the average *Avg.* over all books for each value.

Book	ABBYY	OCR4all	ErrRed.	Impr.
F1781	2.9	0.60	79.3	4.8
F1803	27	4.89	81.9	5.5
F1810	3.8	0.61	84.0	6.2
F1818	10	1.35	86.6	7.5
F1826	1.1	0.06	94.4	18
F1848	0.93	0.20	78.5	4.7
F1851	1.0	0.16	84.0	6.3
F1855	4.0	0.33	91.8	12
F1865	1.6	0.18	88.8	8.9
F1870	0.48	0.13	72.9	3.7
Avg.	**5.3**	**0.85**	**84.2**	**7.8**

4.2.2. Interpretations

ABBYY struggled noticeably with substantially soiled pages, recognizing lines in regions showing dirt or bleed-through on a regular basis, resulting in gibberish OCR output. OCR4all showed only a few segmentation errors, with the main problem being left out page numbers, which happened due to a heuristic in the OCRopus 1 line segmentation script that ignores lines that contain less than three CCs. Table 4 lists the most common OCR errors.

Table 4. The five most common confusions over all ten books for ABBYY Finereader and OCR4all, consisting of the GT, the prediction (OCR), the Counted Number of Occurrences (CNT) and the percent contribution (%) of a given confusion to the overall number of errors. Whitespaces are shown as ␣, and empty cells denote no prediction. Please note that the absolute occurrences of the top error are almost identical for both approaches, but the percentages differ considerably due to the vastly smaller number of errors achieved by OCR4all.

ABBYY				OCR4all			
GT	**OCR**	**CNT**	**%**	**GT**	**OCR**	**CNT**	**%**
␣		64	2.6	␣		63	11.9
	␣	57	2.3	n	u	14	2.7
s	S	57	2.3	f	s	12	2.3
	,	50	2.0	i	l	12	2.3
e	c	40	1.6	r	t	12	2.3
	Remaining		89.2		Remaining		78.5

First of all, it was apparent that the error distribution of the results produced by OCR4all was more top heavy, with the top five making up for almost 22% of the total errors, compared to the one of ABBYY (close to 11%). However, the distributions were actually quite similar to each other, apart from the top error of OCR4all, namely the deletion of whitespaces, which was responsible for almost 12% of the errors alone. Interestingly, while insertions and deletions of whitespaces represented the top two errors for ABBYY and OCR4all also failed to predict them on a regular basis, insertions of whitespaces did not occur in the top ten of OCR4all at all. The remainder of the most frequent OCR4all errors looked as expected, containing well known errors like the confusion of similar looking (at least in 19th Century Fraktur script) characters like *n* and *u*, *f* and *s* (originally predicted as the long s and then regularized), or *c* and *e*, as well as the insertions and deletions of tiny elements like commata, sometimes also as part of quotation marks. Furthermore, some of the aforementioned typical OCR errors like the confusions of *e* and *c* and *s* and *S* still were surprising since one would expect the powerful dictionary and language modeling capabilities of ABBYY to deal with these errors quite comfortably. A possible explanation is that these postcorrection operations did not change characters that have been recognized with a certain degree of confidence to prevent the introduction of errors when "improving" out of dictionary words like unusual proper names.

Again, it has to be emphasized that these very low CERs can only be achieved when a highly performant mixed model is available. In this case, we were able to rely on a strong voting ensemble perfectly fitting the evaluation material. Unfortunately, comparable ensembles are not available for other scripts and languages, yet.

4.3. Precise Segmentation and Trained OCR of Early Printed Books

In this first evaluation, we will examine the performance of OCR4all on the task, which represents its main area of focus: the OCR of early printed books with the aspiration to obtain a (close to) perfect result, both regarding segmentation and OCR, even if this means a substantial amount of manual work for the user.

4.3.1. General Processing Approach

Each book is always processed by a single user with the exception of 1494, which was cut in half and assigned to two users for independent processing. Clear guidelines for the segmentation of any book had been specified beforehand, most notably the very high expected degree of semantic classification of layout elements, the separate tracking of processing times required for segmentation and textual GT production, and the stopping criteria of the ITA, which was reaching a CER of 1% or below.

4.3.2. Overall Time Expenditure and OCR Accuracy

In this first experiment, we evaluated the two main criteria for a workflow with considerable human interaction: the time that had to be invested both for obtaining a sufficient result regarding segmentation and OCR, as well as the achieved OCR accuracy. These criteria were heavily influenced by several factors, which have to be taken into consideration. First, the experience of the user: An experienced user can be expected to be more efficient, both during the segmentation and the GT production phase. In our experiments, we differentiated between two groups of users: On the one hand, there were several first time users with no experience with OCR4all and no or next to no experience with other OCR related tools and processes. On the other hand, we had users with a solid general OCR background and an extensive history of using OCR4all, LAREX, and various transcription tools. In the following, we assign the labels 1 and 2 to the users of the respective groups, with a higher number indicating a more experienced user. To distinguish the individual users from each group, we assign additional labels (A, B, etc.). With the exception of one experienced user (digital humanities), all users were classical humanities scholars. Before starting to work on their respective books on their own, all participants were introduced to the tool by one of the experienced users.

Second, challenges due to the book: Different books can vary considerably, mainly regarding the number of pages, the complexity of the layout, but also the print or scan quality and overall state of preservation. Since the books utilized during our experiments did not show large discrepancies concerning the latter criteria, we just provide the number of pages and distinct semantic layout classes. Table 5 sums up the results.

Table 5. Results of the precise segmentation and trained OCR of early printed books. The books (Camerarius, Narrenschiff, and practical course) are grouped by the experience of the processing users, first the inexperienced (1), then the experienced ones (2). Regarding the segmentation, we provide the number of Pages (#P) and semantic Region types (#R) that had to be distinguished, as well as the time required for the entire book ($t_{\text{Seg.}}$) and per page ($t_{\text{Seg.}}/P$) on average. For the OCR, we indicate the maximum number of GT Lines (#L), which was used to train the final OCR model along with the achieved CER. Furthermore, the time required to correct all lines required to train the final model and one line on average is shown ($t_{\text{Corr.}}$). Finally, some key figures are derived to ensure the comparability of the works. The overall manual time expenditure is calculated for the entire book (t_{All}) by adding up the overall time required for segmentation and OCR and for an average single page (t_{All}/P) by dividing by the number of pages. For both user groups, averaged values of all books (*Mean*) and the corresponding standard deviation (*StdDev.*) are provided if sensible.

Book Short	User Exp.	Segmentation				OCR				Key Figures	
		#P	#R	$t_{\text{Seg.}}$ (min)	$t_{\text{Seg./P}}$ (min)	#L	B (%)	$t_{\text{Corr.}}$ (min)	$t_{\text{Corr.}}/L$ (s)	t_{All} (min)	t_{All}/P (min)
C1532a	1A	55	7	90	1.6	829	0.47	280	20	370	6.7
C1532b	1A	130	7	110	0.8	611	0.73	146	14	256	2.0
C1533	1A	57	5	82	1.4	806	0.20	129	10	211	3.7
C1535	1A	96	7	104	1.1	723	0.39	176	15	280	2.9
C1552	1A	180	6	110	0.6	384	0.20	44	7	154	0.9
C1554	1B	81	6	66	0.8	487	0.36	76	9	142	1.8
C1557	1B	168	5	194	1.2	1342	0.34	187	8	381	2.3
C1558	1A	94	8	139	1.5	751	0.25	183	15	322	3.4
C1561	1B	344	5	275	0.8	395	0.40	48	7	323	0.9
C1563	1C	158	5	140	0.9	1175	0.60	95	5	235	1.5
C1566	1D	471	7	370	0.8	596	0.61	48	5	418	0.9
C1568	1B	342	5	223	0.7	406	0.24	36	5	259	0.8
N1494	1E	156	7	210	1.3	2302	0.69	315	8	525	3.4
N1494	1F	157	7	360	2.3	969	0.82	97	6	457	2.9
N1549	1G	328	7	210	0.6	2824	0.45	155	3	365	1.1
P1474	1H	198	4	29	0.1	700	0.90	230	20	259	1.3
P1509	1I	218	5	390	1.8	1501	0.42	310	12	700	3.2
Mean		**190**	**6.1**		**1.1**	**988**	**0.47**		**10**		**2.3**
StdDev.					**0.5**		**0.22**		**5.2**		**1.5**
C1541	2B	439	8	345	0.8	847	0.92	82	6	427	1.0
C1566	2A	240	7	80	0.3	599	0.57	45	4	125	0.5
C1583	2A	606	7	200	0.3	1647	1.00	123	5	323	0.5
C1594	2A	420	8	200	0.5	352	0.50	26	4	226	0.5
C1598	2B	344	8	245	0.7	256	0.45	28	7	273	0.8
N1498	2A	161	6	130	0.8	622	0.30	22	2	152	0.9
N1499	2A	166	7	105	0.6	632	0.12	110	10	215	1.3
N1506	2A	215	8	180	0.8	3161	0.20	-	-	-	-
P1484	2B	372	3	65	0.2	226	0.34	22	6	87	0.2
Mean		**329**	**6.9**		**0.6**	**927**	**0.49**		**5.5**		**0.7**
StdDev.					**0.2**		**0.30**		**2.4**		**0.4**

Results

For the less experienced users, an average segmentation expense of slightly more than one minute per page was recorded with considerable variations among different users. Fortunately, more experienced users can speed up the process considerably, resulting in about 36 seconds per page on average, again with considerable variations depending on the layout complexity of the book. Regarding the time expenditure required for correcting OCR results for GT production, the vast majority of users invested less than ten seconds per line on average. Both user groups achieved almost identical CERs (0.47% and 0.49%) by utilizing a very similar amount of GT (988 and 927 lines). These results enabled us to compare the time expenditure of the users on a more general level by taking the achieved OCR quality out of the equation. Calculating the time required to process a book, both segmenting it and creating enough GT to obtain an average CER of below 0.5%, resulted in just 0.7 min per page for the experienced users. Compared to the 2.3 min achieved by the inexperienced users, this represents a speedup of more than a factor of three.

Interpretation

The times accounted for segmentation clearly showed that performing a precise and fine grained semantic segmentation of early printed books, even when using a comfortable and versatile tool like LAREX, can still amount to several hours of work for a single book; plus the time to generate GT (either from scratch or by correcting an OCR result) and to train an OCR model. Our experiments showed that the total processing time from beginning (image processing, page segmentation) to end (OCR text with less than 0.5% on average) was less than a day for books containing a few hundred pages.

While this is still a far cry from an expectation of "press a button, wait a few seconds, receive the results", a meaningful comparison would be to look at the current practice of manual transcription as a baseline. We did not thoroughly evaluate the manual transcription from scratch, but to get a rough impression, the users 2A and 2B transcribed a small number of pages from their respective books (C1541, P1484, and 1499). Extrapolating the effort for the entire book led to an overall time expenditure of 44 h for C1541, 47 h for P1484, and 130 h for N1499. Our method therefore reduced the working time from a few weeks to a day, plus the additional effort to weed out the remaining OCR errors, if desired.

Next, the results indicated a high fluctuation of efficiency even within the two user groups, especially among the inexperienced users. Out of the eight books that took longer than one minute per page, four were processed by the same user (1A). The results of N1494 were especially eye-catching since the segmentation took the second user (1F) over 75% longer than the first one (1E) despite both of them working on almost identical material.

Regarding the OCR correction, it is noteworthy that four out of six books that required more than ten seconds per line were processed by a single user, the same that also achieved most of the slow segmentation results (1A). Not only because of the fact that it was the most experienced user (2A) who achieved the worst OCR results of all books, we have no reason to believe that the user had a noteworthy influence on the OCR accuracy. Most importantly, the reachable CER depends on the book and the contained typography, as well as the amount of GT used for training. The obtained results underline this assumption almost perfectly.

While the discussed key figures are very helpful to obtain an overall impression of the amount of manual effort required to process early printed books with OCR4all, further experiments are required to get a deeper understanding of the effects of the ITA and the influence of segmentation guidelines.

4.3.3. Evaluating the Iterative Training Approach

The manual correction effort not only scaled with the number of lines that have to be corrected, but also with their recognition quality. To be able to evaluate the effects and benefits of the iterative training thoroughly, many different values and results were recorded. Since their evaluation and

interpretation is a quite complex task, we first introduce them and describe them in detail in Figure 9 before we list the results of selected works in Table 6.

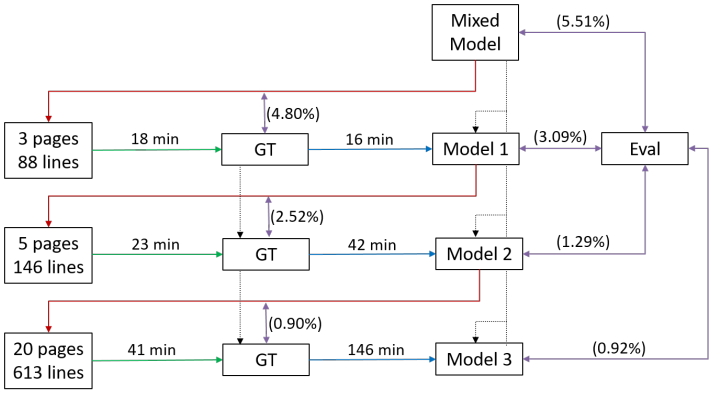

Figure 9. Schematic representation of the ITA and its evaluation. As an example, we used book C1541 processed by an experienced user. For comparison, we refer to the first line of Table 6. To begin with, the user selects a few pages (here, three pages comprising 88 lines) and applies a suitable mixed model to it. After investing 18 min to correct the results, a first evaluation shows that the mixed model achieved a CER of 4.80% on the first batch of lines. Next, the produced GT can be used to train a first book-specific model (Model 1), which required 16 min, using the initial mixed model as a starting point. Model 1 is then applied to the next batch (5 pages/146 lines). After correcting the erroneous results (23 min, 2.52% CER), a second book-specific model is trained (Model 2, 42 min) using all available GT (eight pages/234 lines) and again building from the initial mixed model. This process is repeated until a satisfactory CER is reached or the entire book is transcribed. For evaluation purposes, a separate Eval dataset can be utilized, which was not part of any training set. By applying the mixed model and the models produced during each iteration to this dataset and evaluating the results, we can compare the models objectively.

Table 6. Evaluation of the ITA. For each book processed by a user, we provide all values and results necessary to reconstruct and evaluate the progress of the GT production and training. In the new data column, the number of the newly added Pages (#P) and the corresponding number of Lines (#L) is listed, as well as the time required to produce the transcription. Furthermore, the CER is given, which is calculated from the OCR result achieved by the model from the previous iteration and the newly created GT. For comparison, the CER achieved on the separate and constant Evaluation set (Eval) is recorded. All data shows the number of available GT pages and lines at this point which then serve as training data for the new model, which is used for the next iteration. In the correction columns, we compare the actual required correction time when applying the ITA (ITA) with the projected time when only using the output of the Mixed Model (MM) to get to the point where the final (and in case of MM, the first) model is trained. The Speedup factor (SU) is calculated for each book for the two user groups separately and for both groups combined.

Book Short	User Exp.	It.	New Data					Eval	All Data		Correction		
			#P	#L	$t_{Corr.}$ (min)	$t_{Corr.}$ (s/L)	CER (%)	CER (%)	#P	#L	ITA (min)	MM (min)	SU
C1541	2B	1	3	88	18	12	4.80	5.51	3	88			
		2	5	146	23	9.5	2.52	3.09	8	234			
		3	20	613	41	4.0	0.90	1.29	28	847			
		4	-	-	-	-	-	0.92	-	-	82	169	2.1
P1484	2B	1	5	110	14	7.6	3.53	3.95	5	110			
		2	6	116	8	4.1	0.89	1.48	11	226			
		3	-	-	-	-	-	0.34	-	-	22	29	1.3
N1499	2A	1	2	105	65	37	25.22	23.59	2	105			
		2	3	138	20	8.7	0.54	2.23	5	243			
		3	5	389	25	3.9	1.24	1.63	10	632			
		4	-	-	-	-	-	0.20	-	-	110	474	3.6
												Mean(2):	**2.3**

Table 6. *Cont.*

Book Short	User Exp.	It.	New Data					Eval	All Data		Correction		
			#P	#L	$t_{Corr.}$ (min)	$t_{Corr.}$ (s/L)	CER (%)	CER (%)	#P	#L	ITA (min)	MM (min)	SU
C1557	2B	1	4	104	16	9.2	2.00	10.00	4	104			
		2	11	307	70	14	6.06	8.64	15	411			
		3	15	407	56	8.3	1.60	1.17	30	818			
		4	20	524	45	5.2	0.26	0.65	50	1,342			
		5	-	-	-	-	-	0.34	-	-	187	206	1.1
C1558	1A	1	4	125	38	18	15.31	16.86	4	125			
		2	8	251	60	14	1.28	0.65	12	376			
		3	12	375	85	14	0.58	0.34	24	751			
		4	-	-	-	-	-	0.25	-	-	183	225	1.2
C1566	1D	1	5	122	15	7.4	3.85	4.27	5	122			
		2	6	126	18	8.6	3.15	1.45	11	248			
		3	12	348	15	2.6	0.22	0.99	23	596			
		4	-	-	-	-	-	0.61	-	-	48	74	1.5
												Mean(1):	**1.3**
												Mean:	**1.9**

Results

There were several interesting things to be taken away from the results summarized in Table 6. First of all, it is shown that the ITA yielded a significant speedup regarding the correction time. On average, the manual effort was almost cut in half (average speedup factor 1.9, last column), with the experienced users benefiting considerably more compared to the inexperienced ones (factors of 2.3 and 1.3).

Another eye-catching abnormality was the discrepancies between the performances of the same models on the new and the evaldata. While some deviations had to be expected and can be considered negligible, others seem to be too substantial to be disregarded as variance. For example, when processing, C1557 achieved a good CER of 2% on the new data, but at the same time struggled severely with the eval data (10% CER). An explanation is given in the next section.

Interpretations

Admittedly, the projection of the speedup achieved by the ITA was quite rough since the factor depended much on the pages the mixed model was applied to, which is also shown by the high fluctuations among the speedup factors. Moreover, in a real-world application scenario, there has to be some kind of training and testing during the correction phase in order to know when to stop, as the results from Table 5 showed that the number of lines needed to reach a certain CER varied considerably. Figure 10 depicts this problem and graphically explains the gain obtained by ITA.

Determining the ideal training route is no trivial task and depends on several factors. To begin with, the user has to estimate how many lines are necessary to reach the desired CER. Due to the variety of the material, this is very challenging, even for experienced users, resulting in over- and under-estimations of the required amount. The smaller the chosen steps there are, the more accurate the convergence to the optimal value P (Figure 10) becomes. One (theoretical) approach is training a model each time a new line of GT is added (red curve, left); however, this is not sensible. The other end of the spectrum is represented by correcting the output of the mixed model until the presumably required number of GT lines is reached (green), which discards the gain of correcting lines with an improving CER (area ratio on the right). Consequently, the optimal or rather a sufficient real-world solution has to lie somewhere in between these two extremes. The available hardware plays an important role as it directly influences the training duration. For example, most training processes can be completed within a couple of minutes when using several GPUs, allowing the user to continue the transcription almost instantly. When no GPU support is available, a training can take several hours, requiring the user to perform different tasks.

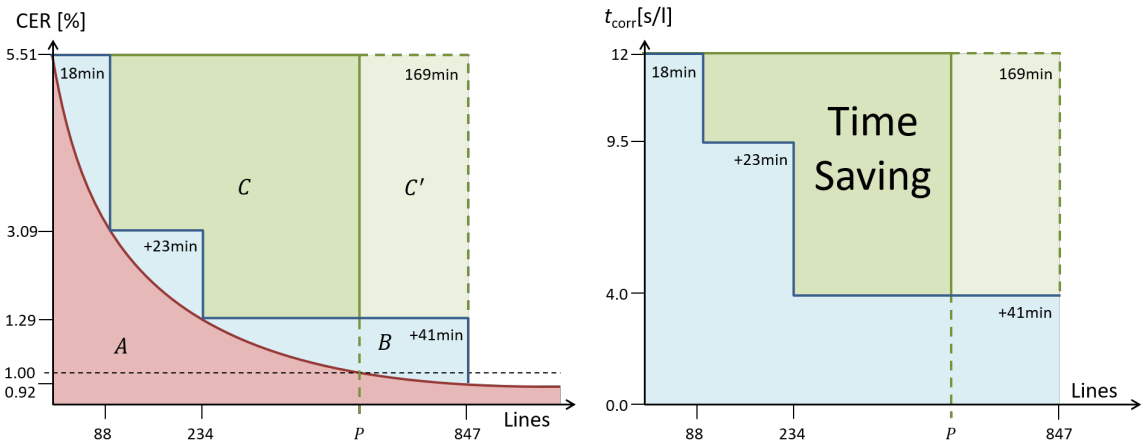

Figure 10. Left: Analysis of the ITA using C1541. The goal is to reach point P, which represents the (unknown) number of lines necessary to reach a CER of below 1%. In theory, the red curve describes the unknown relation between GT lines used for training and the achieved CER and therefore represents the theoretical ideal ITA that produces and applies a new OCR model after the transcription of a single new line of GT. The real route chosen by User 2B is shown by the blue stair case function. Green depicts a single training approach using the same final number of lines of GT as User 2B ($C + C'$) and the perfect, but unknown number of lines (C). Right: This time, the t_{corr}-coordinate shows the manual effort necessary to correct a single error (which highly depends on the CER). The time saved by the ITA (in the case of User 2B and theoretically ideal) is represented by the green area minus the blue area. In this case, this represents a reduction of the manual effort by 51%, which equates to a speedup of 2.1.

Despite the complexity of optimizing the ITA, its general benefits are clear, and the results confirm the expectations. This speedup is due to the fact that the average correction time per line clearly correlates with the quality (CER) of the underlying OCR result. The only exception can be seen in Iterations 1 and 2 of book C1566 where it took the user even a little longer to correct a line, despite starting from a somewhat better recognition result. Since a visual inspection of the concerned pages did not lead to new insights, it stands to reason that human factors like tiredness, form on the day, etc., play a non-negligible role.

A noteworthy observation is the at times striking difference between the CER on the new and the eval data (see the first iteration of C1557 for an example), which can be explained by the unbalanced occurrence of an additional font (italics) in the new and eval data.

N1499 is another interesting case as, despite the rather normal looking Bastarda font and being among the best books in the corpus in terms of image quality, the CER obtained from applying a mixed model was by far the worst that occurred during our experiments, yielding a CER of around 25%. Furthermore, while the recognition quality on the new data in the second iteration was great (0.54%), the resulting model performed significantly worse on the new data of the next iteration (1.24%). This can be attributed to the introduction of marginalia consisting of reference numberings printed in a different type, which led to the introduction of previously negligible errors, all related to the characters *x*, *v*, and *j*, which had a big impact on the CER, but not on the correction time, since they were easy to spot and to fix.

Concerning the training duration (machine time without human intervention), we do not want to go into detail in this paper, as the required times considerably depend on many factors including the available hardware, the amount of available GT, many training parameters, especially the use of data augmentation, and the activation of early stopping. In our experience, a modern PC or laptop is enough to perform standard training runs quickly within one to two hours, while even extensive book-specific training processes can be completed over night. During the course of our experiments, we set up an instance of OCR4all on a server where the data could also be accessed by a highly performant GPU cluster allowing completing most of the training processes in a couple of minutes.

4.3.4. Segmentation without Semantic Classification

As our first experiment has shown, the segmentation step can be considerably more time consuming than OCR, even when aiming for very low CERs. However, the required manual work to segment a book can be severely cut down when the aspirations regarding semantic classification are less strict. Therefore, we conducted another experiment where the single goal of the segmentation was to provide the subsequent OCR with the means sufficient to produce the required output. Apart from a clean text/non-text separation, this also includes ensuring the correct reading order. Figure 11 shows the desired results.

Figure 11. Representative example image of two books showing the difference between the basic (from left to right: 1,3) and exact (2,4) segmentation approach for books C1541 (1,2) and N1506 (3,4).

We selected three books due to their widely differing layout properties: P1484, C1541, and N1506. For our experiment, the users 1C and 2B both segmented 20 representative pages of each book twice, first using the basic approach and then the complex approach from the first experiment.

As expected, the basic segmentation approach required considerably less time than the exact one, leading to an average savings of 38% for the experienced and 45% for the inexperienced user. Regarding the comparison of two users with a different degree of experience, the results in general showed the expected tendency, namely a much faster processing by the more experienced user. On average, it took the novice user 2.65 times longer to perform the basic segmentation and 2.95 longer when an exact segmentation was required. Naturally, the achievable speed-up correlates heavily with the layout complexity.

It is worth mentioning that applying the basic segmentation does not influence the quality of the generated GT in a negative way, since we focus on training a generic model for the task of text/non-text separation. In our experience, a fine-grained semantic distinction of layout elements does not seem feasible due to the high variations between books.

5. Discussion

Before discussing the results and their meaning in detail, we sum up the main findings of our experiments:

- The experiments with 19th Century Fraktur novels showed that a fully automated application of OCR4all is not only possible, but can be highly precise on material with moderate complex layouts and if a suitable OCR model is available (average CER of 0.85% compared to ABBYY's 5.3%).
- When dealing with challenging early printed books, inexperienced users had to invest 2.3 min per page on average to perform a precise segmentation and to reach a CER of below 0.5%, which highlights the effectiveness of the proposed approach. Experienced users can perform much more efficiently, reaching a speedup factor of more than 3 (0.7 min per page on average).
- The ITA yielded significant speedups (factor 1.9) compared to the naive correction of the output of the mixed model.

- A basic segmentation approach that only ensures a sufficient text/non-text separation and a correct reading order reduces the manual effort required for segmentation considerably by a factor of more than 2.5.

The obtained results showed that OCR4all fulfilled its purpose by enabling non-technical users to capture even the earliest printed books completely by their own and with great quality, despite the challenges provided by complex layout and irregular typography. Due to our strict demands regarding the semantic classification of layout elements and our goal of high OCR quality, a considerable amount of manual work was required and accepted. While the experiments showed that even non-technical users without any background or previous experience in OCR were comfortably able to successfully work with OCR4all, the results also showed that there is a learning curve and that experience is key. This holds true for both the segmentation, as well as the OCR, with experienced users being almost twice as fast when it comes to segmenting a page or transcribing a text line compared to inexperienced users on average. However, the quality of the result was not influenced by the experience of the user, with both groups achieving an excellent average CER of slightly below 0.5%.

Regarding the two main steps that require manual intervention, segmentation, and OCR, the first one seems to show more room for improvement since the OCR of historical printings has made great progress over the last few years, which could also be observed during our experiments. Calamari's training and recognition capabilities combined with the easy-to-use ITA provided by OCR4all allow the users to utilize state-of-the-art deep learning software and accuracy improving techniques like pretraining, voting, and data augmentation without ever being forced to acquire a deeper understanding of the technical concepts behind them. As shown by the evaluations, CERs below 1% or even 0.5% should almost be considered the norm after a thorough book-specific training was performed. The segmentation using LAREX proved to be intuitive and highly accurate.

While the usage of OCR4all reduced the manual effort necessary to transcribe early printed books tremendously, especially compared to the fully manual approach, which often required several weeks of full-time transcribing to process a single book, we still think that there is room for improvement.

While a fully automated approach where it is not necessary to look at every single page seems to be currently out of reach, not only due to the complexity of the layouts, but also due to the very high demands of the users regarding the quality of the segmentation and degree of semantic distinction, we think that the efficiency of this part of the workflow can be further increased.

Our evaluations showed that a basic segmentation approach that simply ensures a proper text/non-text separation and a correct reading order can save around 40% of the time required for segmentation. While this represents a substantial speedup, the required manual effort is probably still too high for some areas of application. Yet, we have seen that the segmentation approaches currently available in OCR4all cannot deal with more complex layouts in a (close to) fully automatic manner.

Something that is clearly missing is a fully automatic way to deal with more complex layouts. While deep learning based approaches, mostly aiming to assign a layout able to each individual pixel, carry some promise, they are usually geared towards the training on and application to a single book (see for example [21]). However, when a basic segmentation (text/non-text separation and maybe a correct reading order) is considered sufficient, a mixed models approach for segmentation similar to the OCR one represents a very promising idea. Unfortunately, as of now, we are missing the required GT (page images and pixel labeling) to train these models. Admittedly, there is some fitting data more or less openly available, but usually, they show considerable shortcomings in terms of quality and accuracy, or only offer a manageable amount of data from a small number of books and/or a very specific time period. A comprehensive and versatile dataset of high quality segmentation GT is missing, especially for early printed books.

The current semi-automatic and precise segmentation approach of OCR4all allows producing a large amount of fitting GT with manageable effort. During the evaluations for this paper alone, more than 6000 pages of first-rate GT have been created. Naturally, the manual part of the segmentation represents a non-negligible extra effort and expense, but it also offers a double benefit, as it not only

leads to higher OCR results for the (humanist) users, but it also allows the developers to utilize the new GT to train stronger mixed models for the segmentation task. Of course, the next step then would be to introduce an ITA for the segmentation task to OCR4all, allowing the users to build from a reasonable result produced by a mixed model and then gear it towards the book at hand by correcting a few pages (hopefully) with minimal effort and then train a new model, which might then even be able to provide a more sophisticated semantic distinction of layout element if desired by the user and properly trained.

When discussing the means necessary to increase the degree of automation, the need for a book-specific training also plays an important role. While our experiments have confirmed the effectiveness and efficiency of the ITA, it still represents a time consuming task. Despite our focus on projects intending to produce quotable text as their final result, we are aware of the fact that there are other use cases that, despite dealing with early printed books, aim for a more quantitative approach and therefore are willing to make sacrifices regarding text quality and semantic labeling. As we have seen above, despite our repertoire of mixed models trained on a wide variety of fonts geared towards the recognition of different font classes, a sufficient OCR quality, for example 95%+, cannot be guaranteed at all when working with early printed books. Of course, a wider variety of more specialized mixed models can help to improve the results.

Just like with the segmentation task, the precise book-specific processing also creates a great deal of valuable OCR GT (over 25,000 lines during our experiments). Again, these data will be utilized for the continuous refinement of existing mixed models, basically resulting in a more broadly arranged ITA aiming to constantly reduce the required manual effort by optimizing the starting points for each book specific training.

Another challenging task is the mixture of several fonts or even scripts within a book, while even a mixture on a single page or even within a line is not uncommon, as we have seen during our Camerarius use case. We already showed Calamari's ability to reliably and accurately distinguish between different fonts even when their typefaces were very similar to each other [22]. Kraken [23] has also taken a few first steps in that direction by providing a generic model that can differentiate between Latin, Arabic, Greek, and Syriac script. After detecting the script, suitable OCR models can be applied to the (parts of) lines that match.

In general, the applicability of fully automated methods does not only depend on the intended usage of the results, but also on the material at hand. We have shown that OCR4all can achieve excellent results on 19th Century Fraktur novels with a moderate layout and a suitable mixed OCR model available. As expected, the fully automatic processing of early printed books is a tricky task, and its applicability also highly depends on factors like layout and typography. The first experiments led to the following mostly qualitative observations:

- The current setup can deal with relatively simple layouts consisting of a single or several well separated columns quite reliably. When several columns have to be identified, the user needs to specify the maximum number of columns occurring on a page.
- Despite the lack of an explicit text/non-text segmentation, the combination of OCRopus 1 line segmentation and Calamari's recognition module is surprisingly robust against non-text elements like noise, artistic border elements, images, and swash capitals. Even if parts of these elements make it into a text line, they often do not deteriorate the text recognition result since Calamari will ignore them due to the lack of a confident recognition of available characters.
- Marginalia that are located very close to the main text often cannot get separated correctly, leading to significant errors in the reading order.
- Treating a page that comprises highly varying font sizes, for example a very prominent heading line and many running text lines whose characters are not even half as high, as a single text segment can lead to wrongly segmented lines. This happens because the line segmentation estimates the most likely height of a line on page level and then tries to find fitting lines. A preceding region segmentation prevents this problem from occurring.

- The available mixed models work reasonably well on the majority of books, achieving an average CER of 7.7% on the corpus we used for our evaluations. However, since this is probably not good enough for most use cases, book specific training is necessary. Additionally, the CERs varied considerably, ranging from below 2% to over 25% on the new GT of the first iteration of each book, underlining the problem of mixed models we discussed before.

To sum up, despite the open questions and challenges demonstrated above, OCR4all can become a cornerstone when it comes to the high quality capturing of historical printings (a final summary of the obtained results is shown in Figure 12). By reducing the required technical know how to a minimum, it is now possible for humanities scholars to take the acquisition of their much desired and needed textual research data into their own hands.

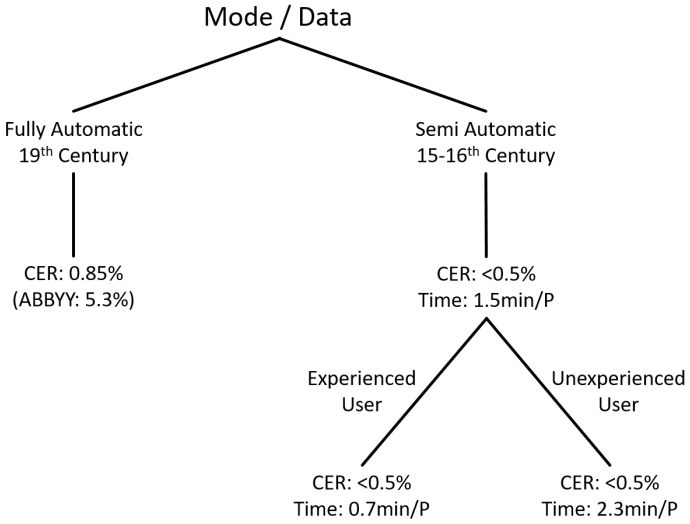

Figure 12. Final summary of the results dependent on the material and the processing user.

6. Future Work

In this section, we first discuss features we would like to integrate into OCR4all or its submodules in the future before concluding the paper by giving an outlook on the general future of OCR4all.

First of all, one of the main goals to address is the obvious lack of a more potent segmentation method that can deal with (more) complex layouts in a highly automized way. Therefore, we aim to include a trainable pixel classifier in order to either provide a valid starting point for other segmentation approaches by classifying pixels and consequently connected contours as text, image, and noise or even perform a fine-grained semantic markup. Of course, a more powerful segmentation approach must also comprise a more sophisticated method for the determination of the reading order, which also has to be integrated into LAREX. To generate the reading order, the idea is to allow the user to specify comfortably rules based on the detected region types, as well as their absolute and relative position.

Regarding training and recognition, we want to provide the user with the possibility to train comfortably several type-specific models for a single work. This can be very helpful, when the book comprises a few, possibly highly differing, fonts or even scripts, e.g., Latin and Greek. A book-specific trained script detection model in fact should be expected to perform considerably better, not to mention that it is not feasible to train generic mixed models to recognize book-specific fonts. Consequently, an appropriate GT production and training functionality will have to be supported within OCR4all.

Furthermore, in order to train more robust models, a more flexible selection of lines for training, recognition, and correction is desirable as this allows training models using GT that is widely spread over the course of the entire book. This would help to further optimize the ITA by integrating active learning, i.e., adding lines to the existing GT pool, which the current models had problems recognizing, instead of random ones. The effectiveness of this approach has already been shown by [24], who

were able to improve considerably the OCR results (average gain of 16%, maximum gain of 32%) by purposefully adding lines to the training set that showed the largest disagreement between the separate outputs of the voters.

Apart from the voting, there are several other use cases for which we want to profit from the character confidences provided by Calamari. First, the correction process can be supported by highlighting suspicious characters. Second, by averaging the confidence values over several lines, it is possible to identify segments or pages that contain a worse recognition result compared to the rest of the book. This could help to identify text parts that suffer from an increased amount of degradation, contain segmentation errors, use a different type of font, etc. Third, the average confidence calculated over a representative number of recognized lines can serve as a form of quality estimation. We know that the confidence values correlate with the recognition rate and that the neural networks tend to overestimate their performance. Therefore, we hope that it is possible to use many existing measurements to derive a model that is able to estimate the true recognition accuracy based on the average confidence. In addition to the automatic selection of the best fitting model for given data, this would be particularly helpful when the goal of a OCR process is to reach a certain recognition quality (for example, a 2% CER is sufficient for most NLP tasks), and it is unclear whether the output of a mixed model suffices or if book-specific training is required.

A most desirable issue that has to be addressed as soon as possible is the incorporation of a post-processing step, for example using dictionaries or language modeling. However, especially for (very) early printed books, this is not a trivial task due to the lack of consistent spelling rules and the frequent use of abbreviations.

A particularly interesting and challenging goal is to overcome the additional difficulties of handwritten recognition. Despite there being several steps that would require adaptations, for example the line segmentation, the remaining steps work quite similarly to when dealing with printed texts. Actually, OCR4all has already been successfully applied to a Greek manuscript (Aëtius Amidenus—Libri medicinales, 16th Century), achieving character recognition rates in the mid-nineties when using only a few hundred lines of GT.

As mentioned above, OCR4all's primary field of application was planned to be the local setup at a single user desktop PC or laptop. However, with some manageable extensions regarding a project and user administration system, as well as an interface to a resource scheduling manager, OCR4all can be deployed and run as a full-featured web service. This would be especially helpful for institutions or working groups who want to share their resources among themselves in order to work collaboratively. Even without further extensions, a collaborative approach is already possible: During our experiments, we set up an instance for several users to cooperate in a somewhat coordinated way, which proved to be highly effective.

Further, smaller goals are the integration of the scan preparation step directly into the web GUI, enabling a TEI output by implementing sensible and configurable mapping from PageXML to TEI, and a confidence based fuzzy search that allows finding words even when they contain OCR errors.

A key aspect remains the optimization of the teaching material associated with the tool. In the future, we want to build from the already existing written guides not only by adding screencasts or even tutorial videos, but by setting up a knowledge base, most likely in the form of a Semantic MediaWiki that not only contains the guides mentioned above in a more modular form, but also extends them with and crosslinks them to the theoretical concepts behind each individual step of the OCR4all workflow. Combined with a public repository for GT and models, best practices, as well as an assembly of frequently occurring difficulties, and proven ways to successfully deal with them, this would provide the community with a place to share material, knowledge, problems, and solutions.

Despite these comprehensive plans for the future, we already reached our main goal of creating a tool that provides non-technical users with access to a powerful and easy-to-use OCR workflow. This is not only shown by the evaluations, but also by the successful application in numerous real-world projects where OCR4all leads to significant speedups of the OCR of our precious cultural heritage.

Author Contributions: Conceptualization: C.R. and F.P.; methodology: C.R. and U.S.; software: C.R., D.C., A.H., N.B., M.W., C.W., and A.B.; formal analysis: C.R., F.P., and C.W.; investigation, C.R., M.W., N.B., and F.P.; writing, original draft preparation: C.R. and F.P.; writing, review and editing: C.R., F.P., U.S., M.W., C.W., C.G., N.B., A.H., and A.B.; supervision: C.R., F.P., and U.S.; project administration: C.R. and F.P.; funding acquisition: F.P.

Funding: The development of OCR4all was funded by the Chair for Artificial Intelligence and the Centre for Philology and Digitality at the University of Würzburg, as well as the Federal Ministry of Education and Research project Kallimachos. This publication was funded by the German Research Foundation (DFG) and the University of Würzburg in the funding programme Open Access Publishing.

Acknowledgments: The authors would like to express their gratitude to the entire Narragonien digital work group around Brigitte Burrichter and Joachim Hamm who helped to establish, test, and optimize the OCR4all workflow in close cooperation. Furthermore, we would like to thank the Opera Camerarii team around Thomas Baier, Marion Gindhart, Joachim Hamm, and Ulrich Schlegelmilch for providing a valuable and challenging use case and test object for OCR4all. Finally, our gratitude is due to everyone who supported the planning, implementation, evaluation, distribution, and presentation of OCR4all including Sophia Beckenbauer, Kevin Chadbourne, Björn Eyselein, Yannik Herbst, Raphaëlle Jung, Tanja Kohl, Florian Langhanki, Maximilian Nöth, and Ronja Wegrath.

Conflicts of Interest: The authors declare no conflict of interest.

Abbreviations

The following abbreviations are used in this manuscript:

CC	Connected Component
CER	Character Error Rate
CNN	Convolutional Neural Network
CPU	Central Processing Unit
CTC	Connectionist Temporal Classification
FAS	Fully Automatic Segmentation
GPU	Graphical Processing Unit
GT	Ground Truth
ITA	Iterative Training Approach
LSTM	Long Short Term Memory
OCR	Optical Character Recognition

References

1. Doermann, D.; Tombre, K. (Eds.) *Handbook of Document Image Processing and Recognition*; Springer: London, UK, 2014; doi:10.1007/978-0-85729-859-1. [CrossRef]
2. Rydberg-Cox, J.A. Digitizing Latin incunabula: Challenges, methods, and possibilities. *Dig. Hum. Q.* **2009**, *3*. [CrossRef]
3. Springmann, U.; Lüdeling, A. OCR of historical printings with an application to building diachronic corpora: A case study using the RIDGES herbal corpus. *Dig. Hum. Q.* **2017**, *11*, 146–160.
4. Fischer, A.; Wüthrich, M.; Liwicki, M.; Frinken, V.; Bunke, H.; Viehhauser, G.; Stolz, M. Automatic transcription of handwritten medieval documents. In Proceedings of the 15th International Conference on Virtual Systems and Multimedia, 2009 (VSMM'09), Vienna, Austria, 9–12 September 2009; pp. 137–142.
5. Hochreiter, S.; Schmidhuber, J. Long short-term memory. *Neural Comput.* **1997**, *9*, 1735–1780. [CrossRef] [PubMed]
6. Graves, A.; Fernández, S.; Gomez, F.; Schmidhuber, J. Connectionist temporal classification: labelling unsegmented sequence data with recurrent neural networks. In Proceedings of the 23rd International Conference on Machine Learning, Pittsburgh, PA, USA, 25–29 June 2006; ACM: New York, NY, USA, 2006; pp. 369–376.
7. Breuel, T.M.; Ul-Hasan, A.; Al-Azawi, M.A.; Shafait, F. High-Performance OCR for Printed English and Fraktur Using LSTM Networks. In Proceedings of the 12th International Conference on Document Analysis and Recognition, Washington, DC, USA, 25–28 August 2013; pp. 683–687, doi:10.1109/ICDAR.2013.140. [CrossRef]

8. Breuel, T.M. High Performance Text Recognition Using a Hybrid Convolutional-LSTM Implementation. In Proceedings of the IEEE 2017 14th IAPR International Conference on Document Analysis and Recognition (ICDAR), Kyoto, Japan, 9–15 November 2017; pp. 11–16.
9. Wick, C.; Reul, C.; Puppe, F. Comparison of OCR Accuracy on Early Printed Books using the Open Source Engines Calamari and OCRopus. *JLCL Spec. Issue Autom. Text Layout Recognit.* **2018**, *33*, 79–96.
10. Breuel, T.M. The hOCR microformat for OCR workflow and results. In Proceedings of the IEEE Ninth International Conference on Document Analysis and Recognition (ICDAR 2007), Curitiba, Paraná, Brazil, 23–26 September 2007; Volume 2, pp. 1063–1067.
11. Pletschacher, S.; Antonacopoulos, A. The PAGE (page analysis and ground-truth elements) format framework. In Proceedings of the IEEE 2010 20th International Conference on Pattern Recognition, Istanbul, Turkey, 23–26 August 2010; pp. 257–260.
12. Merkel, D. Docker: lightweight linux containers for consistent development and deployment. *Linux J.* **2014**, *2014*, 2.
13. Wick, C.; Reul, C.; Puppe, F. Calamari—A High-Performance Tensorflow-based Deep Learning Package for Optical Character Recognition. *Dig. Hum. Q.* **2018**, forthcoming.
14. Abadi, M.; Barham, P.; Chen, J.; Chen, Z.; Davis, A.; Dean, J.; Devin, M.; Ghemawat, S.; Irving, G.; Isard, M.; et al. Tensorflow: A system for large-scale machine learning. In Proceedings of the 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16), Savannah, GA, USA, 2–4 November 2016; pp. 265–283.
15. Phillips, I. User's reference manual for the UW english/technical document image database III. In *UW-III English/Technical Document Image Database Manual*; Intelligent Systems Laboratory: Seattle, WA, USA, 1996.
16. Reul, C.; Springmann, U.; Wick, C.; Puppe, F. State of the Art Optical Character Recognition of 19th Century Fraktur Scripts using Open Source Engines. In Proceedings of the DHd 2019 Digital Humanities: Multimedial & Multimodal, Mainz, Germany, 25–29 March 2019.
17. Smith, R. An overview of the Tesseract OCR engine. In Proceedings of the IEEE Ninth International Conference on Document Analysis and Recognition (ICDAR 2007), Curitiba, Paraná, Brazil, 23–26 September 2007; Volume 2, pp. 629–633.
18. Breuel, T.M. The OCRopus open source OCR system. In *Document Recognition and Retrieval XV*; International Society for Optics and Photonics: Bellingham, WA, USA, 2008; Volume 6815, p. 68150F.
19. Afzal, M.Z.; Krämer, M.; Bukhari, S.S.; Yousefi, M.R.; Shafait, F.; Breuel, T.M. Robust binarization of stereo and monocular document images using percentile filter. In Proceedings of the International Workshop on Camera-Based Document Analysis and Recognition, Washington, DC, USA, 23 August 2013; Springer: Berlin, Germany, 2013; pp. 139–149.
20. Paszke, A.; Gross, S.; Chintala, S.; Chanan, G.; Yang, E.; DeVito, Z.; Lin, Z.; Desmaison, A.; Antiga, L.; Lerer, A. Automatic Differentiation in PyTorch. In Proceedings of the NIPS Autodiff Workshop, Long Beach, CA, USA, 8 December 2017.
21. Chen, K.; Seuret, M.; Liwicki, M.; Hennebert, J.; Ingold, R. Page segmentation of historical document images with convolutional autoencoders. In Proceedings of the IEEE 2015 13th International Conference on Document Analysis and Recognition (ICDAR), Nancy, France, 23–26 August 2015; pp. 1011–1015.
22. Reul, C.; Göttel, S.; Springmann, U.; Wick, C.; Würzner, K.M.; Puppe, F. Automatic Semantic Text Tagging on Historical Lexica by Combining OCR and Typography Classification. In Proceedings of the 3rd International Conference on Digital Access to Textual Cultural Heritage, Brussels, Belgium, 8–10 May 2019.
23. Kiessling, B. Kraken—An Universal Text Recognizer for the Humanities. In Proceedings of the DH 2019 Digital Humanities: Complexities, Utrecht, The Netherlands, 9–12 July 2019.
24. Reul, C.; Springmann, U.; Wick, C.; Puppe, F. Improving OCR Accuracy on Early Printed Books by combining Pretraining, Voting, and Active Learning. *JLCL Spec. Issue Autom. Text Layout Recognit.* **2018**, *33*, 3–24.