From Historical Newspapers to Machine-Readable Data: The Origami OCR Pipeline

Bernhard Liebl[a], Manuel Burghardt[a]

[a] *Computational Humanities Group, Leipzig University, Leipzig, Germany*

Abstract

While historical newspapers recently have gained a lot of attention in the digital humanities, transforming them into machine-readable data by means of OCR poses some major challenges. In order to address these challenges, we have developed an end-to-end OCR pipeline named *Origami*. This pipeline is part of a current project on the digitization and quantitative analysis of the German newspaper "Berliner Börsen-Zeitung" (BBZ), from 1872 to 1931. The *Origami* pipeline reuses existing open source OCR components and on top offers a new configurable architecture for layout detection, a simple table recognition, a two-stage X-Y cut for reading order detection, and a new robust implementation for document dewarping. In this paper we describe the different stages of the workflow and discuss how they meet the above-mentioned challenges posed by historical newspapers.

Keywords

end-to-end OCR, historical newspapers, layout detection, deep neural networks

1. Introduction

In recent decades a large number of newspapers have been digitized[1], providing access to a unique collection of historiographical data. This opens up a number of opportunities for historical research, but also entails many challenges[2]. A major technical challenge lies in the conversion of scanned newspapers into machine-readable data that can be processed and analyzed in a quantitative way by means of text mining techniques. This conversion is typically referred to as optical character recognition (OCR). However, OCR is a complex topic that involves a number of different processing steps[3]. These steps are particularly challenging for the case of historical newspapers, as they oftentimes have rather low paper quality, use various historical fonts or require the recognition of complex page layouts in order to separate text from images and tables.

In this paper we present the *Origami* OCR pipeline[4], which was developed as part of a current project that aims at processing the Berliner Börsen-Zeitung (BBZ), a German newspaper with

CHR 2020: Workshop on Computational Humanities Research, November 18–20, 2020, Amsterdam, The Netherlands

liebl@informatik.uni-leipzig.de (B. Liebl)

0000-0003-1354-9089 (M. Burghardt)

[1] *Google newspapers* (https://news.google.com/newspapers); *Europeana* collection of newspapers (https://www.europeana.eu/de/collections/topic/18-newspapers); *ZEFYS* Zeitungsinformationssystem der Staatsbibliothek zu Berlin (http://zefys.staatsbibliothek-berlin.de)

[2] The opportunities and challenges of digitized newspapers were recently discussed in a workshop titled "Digitised newspapers - a new Eldorado for historians ?" (https://impresso.github.io/eldorado/).

[3] For an overview, check out the workflows of the OCR-D project [3, 33, 34] or the OCR4all project [40].

[4] Our source code is publicly available on GitHub (https://github.com/poke1024/origami). As this is still partially a work in progress, we appreciate any feedback on *Origami*.

a focus on finance and economics, for the period 1872-1931. *Origami* tries to take into account the specific challenges for digitizing historical newspapers such as the BBZ. At its core, *Origami* is a new end-to-end (i.e. from document scan to *PAGE XML* [37]) pipeline that covers all essential aspects of a text digitization pipeline, such as (1) tools for generating and annotating ground truth, (2) dewarping, (3) segmentation (using state-of-the-art deep neural networks), (4) layout detection (through a configurable architecture offering heuristic layout rules), (5) separator-aware reading order detection, (6) line-level OCR, and (7) exporting to *PAGE XML* [37]. *Origami* uses existing technologies (e.g. *Tesseract* [45] for baseline detection and *Calamari* [52] for line-level OCR), but also contains some new modules previously not available as working implementations. Apart from OCR-D and the closed source products ABBYY Finereader[5] and Transkribus[6], we know of no other framework or pipeline currently available that implements all necessary steps to perform end-to-end OCR for historical newspapers. *Origami* is easy to setup and built for processing hundreds of thousands of documents.

Employing a Calamari-based OCR with a custom model and without using post-correction, *Origami* achieves overall[7] word error rates (WER) of around 2% on typical BBZ pages with complex layouts and both Antiqua and blackletter typefaces. Judging from publicly available data, this seems better than the performance obtainable with the best commercial systems currently available for newspapers in this time period [24, 9].

2. End-to-end OCR Workflows for Historical Newspapers: State of the Art

A review of related research shows that the digitization of historical newspapers is carried out in quite different ways and with quite different objectives [54, 51, 25]. As Wulfman et al. summarize, in practice there are often opposing goals "of emphasizing the quantity or the quality of digital editions" [54]. Along these lines, Smith & Cordell note that "[t]oo often, discussions of OCR quality in digital humanities (and adjacent fields) begin and end with frustration over its imperfections" [44]. Consequently, one option is to accept these imperfections from the outset and try to remedy them later, by means of post-processing (for an example see [55]). However, post-processing is not a viable option when the base quality of the OCR is too low. Naturally, this raises the question of what "too low" actually means, and the answer depends strongly on the specific application scenario [50, 14, 49, 21, 48].

The moving target of "good quality" is also strongly influenced by the available technology at a given time. In 2009, Tanner et al. reported that any text printed "pre-1900 will be fortunate to exceed 85% accuracy (15 in 100 characters wrong)" [48]. Ten years later, the best solutions for such documents achieve a character accuracy of above 99% [9].

The best way to digitize a corpus is associated with even more difficult decisions about software packages and processing workflows [40, 34]. The task of building a high-quality, end-to-end OCR pipeline that produces good results by leveraging recent advances in technology is quite complex. Although there are literally hundreds of OCR tools[8], the number of working, open source tools for building the core parts of high-quality OCR pipelines boils down to probably less than ten established modules. Unfortunately, all of those have their own limitations,

[5] https://www.abbyy.com
[6] https://transkribus.eu
[7] i.e. taking into account correct segmentation and correct reading order
[8] https://www.digitisation.eu/tools-resources/tools-for-text-digitisation/.

such as dependency on document layouts (e.g. no tables, only columns, no columns), issues with image resolution or unsatisfying quality of pretrained font models [34, 40].

Further issues arise when trying to combine these single components to create an end-to-end OCR pipeline. To date, the OCR-D project [3, 33] seems to be the de facto standard when it comes to providing a coherent end-to-end OCR workflow. OCR-D has been gathering, combining and improving the best non-commercial OCR software components from the last decades and integrating them into the OCR-D framework since 2015. Nevertheless, we could not readily use this framework for our historical newspapers. On the one hand, this was due to the limitations of single components (see above). On the other hand, there were problems because of the lack of flexibility to adapt individual components in the workflow or to add new ones. Some practical issues we encountered when trying to use the OCR-D workflow for our historical newspaper include the following[9]:

- *Pretrained font models.* Although Calamari and OCR-D Calamari offer good OCR models for reading 19th-century Fraktur fonts, no pretrained models exist that recognize both Antiqua and German Fraktur[10]. Additional stages that filter regions by font type need to be employed.

- *Dewarping.* OCR-D offers the only modern (DNN-based) dewarping implementation at the page level (i.e. dewarping a whole page vs. dewarping single smaller regions). Unfortunately, this implementation currently seems to break down with higher resolutions (see Section 3.3).

- *Layout detection.* OCR-D offers a curated set of segmentation and layout algorithms, including what seems to be the only public and currently maintained layout detection from ICDAR's 2013 "Competition on Historical Newspaper Layout Analysis" [2]. However, this solution does not seem to be based on the current technical state of the art for image segmentation, namely DNNs [28, 3]. The same is true for other layout detection modules currently offered by OCR-D (i.e. those based on Tesseract and CIS OCR-D[11]). In general, document types that are not simple multi-column layouts currently pose a huge issue for many modules. For example, as described in its documentation, the CIS OCR-D segmentation module will usually fail if there are any tables in the document. On the other hand, while there have been recent advances in DNN-based table detection such as PubTabNet [56] and TableBank [27], publicly available implementations of these approaches are scarce.

- *Monolithic modules.* While the OCR-D workflow as a whole is highly configurable and modular, single OCR-D modules are, for historical reasons, rather monolithic and do not allow for an easy extension, for instance by means of an API. For the case of the BBZ this proved problematic, as – just as one example – we wanted to extend the X-Y cut functionality to detect separator information, which is valuable for determining reading

[9] We would like to emphasize that the following problems could only be observed for the context of the digitization of the BBZ. The applicability of OCR-D for many other scenarios - especially the digitization of the Union Catalogue of Books of the 16th–18th century (VD16, VD17, VD18), for which OCR-D was originally designed, is beyond question.

[10] Also, the models are only trained for binarized input, which is a strategy we find problematic, as our results in Section 3.6 demonstrate.

[11] https://github.com/cisocrgroup/ocrd_cis

orders. However, X-Y cut implementations available through OCR-D are hard-coded and do not allow for custom scoring functions (as, for example, in [31]).

Besides these practical issues with the generic OCR-D framework, we also encountered limitations with specific workflows for the digitization of historical newspapers as they are reported by several projects and institutions: Jerele et al., who digitized historical Slovenian newspapers with ABBYY FineReader[12], report issues with segmentation as being the "second major error inducer" (the first being "bad condition of newspaper originals") [22]. The challenge of segmentation is also underlined by Dannélls et al. [14], who used a combination of ABBYY FineReader and Tesseract to digitize the Swedish newspaper corpus *KubHist*. The BBZ corpus we work with was also originally processed using ABBYY FineReader 11. Probably also as a result of segmentation issues, we found typical word error rates above 10%.

A very recent (May 2020) project for digitizing historical German-Brazilian Newspapers (GBN)[13] is based on a modified OCR-D workflow and is similar to our project in various aspects. For example, GBN seems to use similar region merge operations as *Origami*'s layout stage (see Section 3.4). However, the overall scope of the project is quite different. Although GBN offers segmentation of documents into regions and lines using a DNN[14], GBN's segmentation seems to generate purely binary labels (i.e. only differentiates text from background), and - in contrast to *Origami* - does not know about other content such as tables, illustrations or separators. GBN is thus neither aware of separators, nor does it attempt to detect tables or reading order. Furthermore, GBN offers deskewing, but no dewarping.

Most recently, the *Impresso* project[15], a large project working on making 200 years of historical newspapers digitally available through new, innovative interfaces, used Transkribus HTR for their OCR stage [46], and – as far as we understand it – for the whole OCR workflow (i.e. ground truth annotation, page segmentation and so on). While the results from *Impresso* are very promising [46], we decided against using Transkribus for our project for two reasons. First, neither the exact software nor the performance of Transkribus is very well documented [40]. Reul et al. note: "to the best of our knowledge the exact state of the software actually incorporated in Transkribus is not publicly known" [40]. Transkribus therefore seems like a black box, which contradicts our goal of open sourcing as many components of our work in this project as possible (e.g. the final line-level OCR models). Second, the closed source approach (Transkribus is run as a "fee-based cooperative" [40]) usually prevents projects from "running the advanced recognition tools on their own hardware" [40]. Digitizing a large scanned corpus (a total of 642,480 pages in our case) on third-party services did not seem an appealing option for our project.

All in all, current end-to-end OCR workflows are not appropriate when it comes to the digitization of historical newspapers. Specific approaches for historical newspapers also have some severe limitations that are mostly connected to poor segmentation of the complex page layouts. To address these existing challenges and limitations, we present *Origami* as an open source OCR pipeline that is optimized for the digitization of historical newspapers. Whenever possible, we use working open source solutions (e.g. line-level OCR, baseline detection) and

[12] In addition, Aletheia [12] was used for ground truth production.

[13] https://github.com/sulzbals/gbn

[14] The latter is achieved through a repurposing and extension of the DNN-based SBB textline detector from Q4/2019 (available at https://github.com/qurator-spk/sbb_textline_detection) and therefore partially overlaps with our BBZ-specific DNN work from this period [28].

[15] https://impresso-project.ch/

experiment with new approaches where we found the results of current open source implementations not adequate for our task[16].

3. The *Origami* OCR Pipeline

Our OCR pipeline currently consists of nine subsequent stages that are shown in Table 1. Each stage runs as a batch and saves its data independently as files.

The first two stages divide the page into different regions (e.g. text vs. tables). Stages 3 and 4 then produce a deskewed and dewarped page, where text lines and borders are set straight. Stage 5 refines the regions found in stages 1 and 2 using heuristic rules. Stages 6, 7 and 8 detect lines, reading order, and perform the OCR for each line. Stage 9 produces a final output file that contains all detected information. In the following sections we describe all stages in more detail.

At first glance, this workflow seems similar to established OCR workflows like used in OCR-D. There is one important difference though. Most OCR workflows operate on sets of images. For example, as shown in Figure 2, in OCR-D workflows, page images are usually binarized and cropped before being split into smaller region and line images. The final OCR then uses these line images.

One big advantage of this approach is that batch interfaces and data formats are easy to understand, as they are modularized and clean: there are only images in each step. A considerable disadvantage of this approach, however, is that context information is lost. For example, a batch operating on a line image will have no way of knowing where this line originally came from or what its relation to other lines or regions on the page was. Yet this information can be useful in further steps like layout detection.

One of *Origami*'s main contributions is experimenting with a different, more complex approach in terms of data management. Instead of producing new images with each step, *Origami* batches write knowledge gained about the page into a set of custom, but well-defined file formats[17]. Figure 1 shows the whole pipeline: solid black boxes represent the batches from Table 1, framed non-solid boxes are data files (called *artifacts* in *Origami*), and arrows indicate reading and writing of artifacts. For example, the *segment* stage writes a *zip* file that contains *png* files with the pixelwise page segmentations. Similarly, the *contours* stage writes a *zip* file that contains descriptions of polygonal regions. Subsequent stages read these files and produce new data based on them. The original page image remains untouched during the entire process, i.e. it is not split or subdivided into new image files. In mathematical terms, we store parameters of a function composition, but not the intermediary results themselves.

3.1. Segment (stage 1)

Stage 1 uses a custom-trained deep neural network (DNN) to determine different types of regions and separators on a pixel by pixel basis. For example, a region predictor classifies regions using the labels *TEXT* (i.e. text), *TABULAR* (tables and tabular multi-column structures inside body text), *ILLUSTRATION* (borders, images and other illustrations) and *BACKGROUND* (background including dirt and frame borders). Similarly, a separator predictor differentiates H (horizontal), V (vertical), and T (table column) separators.

[16]These include dewarping, layout and reading order detection, binarization, and available Fraktur models.

[17]Details of the file formats are documented at https://github.com/poke1024/origami/blob/master/docs/formats.md.

Table 1
Processing stages of the *Origami* pipeline.

	stage name	description
1	segment	separates text, tables, and illustrations (pixel by pixel)
2	contours	finds and simplifies polygonal forms for regions and separators
3	flow	estimates skew and warp at various points of the page
4	dewarp	computes a global dewarping transformation
5	layout	merges and splits regions using heuristic rules
6	lines	detects baselines for all regions
7	order	determines reading order using X-Y cuts
8	ocr	runs OCR on line level using Calamari
9	compose	combines all data into one final document

The specific setup we use and describe here follows the optimal configuration found in a previous evaluation study [28]. In contrast to recent hybrid approaches like Barman et al. [5] - who combine a segmentation DNN with separate OCR information to produce segmentation data - we use a single, but complex DNN that is trained to detect all relevant classes (we expect this network to learn some OCR features as well). In contrast to dhSegment [35] - which uses a variant of ResNet-50 - we use the larger Inception-ResNet-v2 [6] for region segmentation (trained with categorical cross-entropy loss) [28]. For separator detection, we use EfficientNet-B2 (trained with generalized dice loss) [28]. Both our models employ Feature Pyramid Networks (FPNs) as a segmentation architecture.

Before passing data to the DNN, we rescale pages to a total resolution of 1280×2400 pixels, then run each network on three overlapping vertical tiles (each having a resolution of 1280×896 pixels). These resolutions are rather high compared to standard dhSegment workflows [35]. We showed that this configuration reaches a pixel accuracy of 98.86% for regions and 99.79% for separators [28]. For each task, we use confidence voting over models trained from five folds.

Figure 3 shows the output of the neural networks for a typical page from the BBZ. Among other classes, our DNN differentiates between plain vertical separators that occur between regions of body text (V), and vertical lines that delimit columns in tables (T).

3.2. Contours (stage 2)

In this stage, we convert the pixel-based segmentation data from the previous step into polygonal forms and simplify it (this process is sometimes also referred to as *vectorization*). We do this for two reasons: (1) to increase the performance of subsequent pipeline stages, since processing simplified vector shapes is often much faster than looking at a high number of pixels. (2) To be able to use established geometric algorithms in later stages of the pipeline (e.g. Voronoi diagrams), many of which are usually formulated and implemented in terms of polygonal data. Having found the contours, this step also employs a simple heuristic to remove textual noise. We remove all contours that are narrower than a pre-defined threshold and lie on the left or right border.

The determination of region contours is a standard problem of computer vision [47]. We use OpenCV [7] for this task. Extracting exact inner polylines from thick separator pixel groups on the other hand poses a more difficult problem and good algorithms are hard to find. Though this might seem a marginal problem at first, the same challenge occurs when extracting

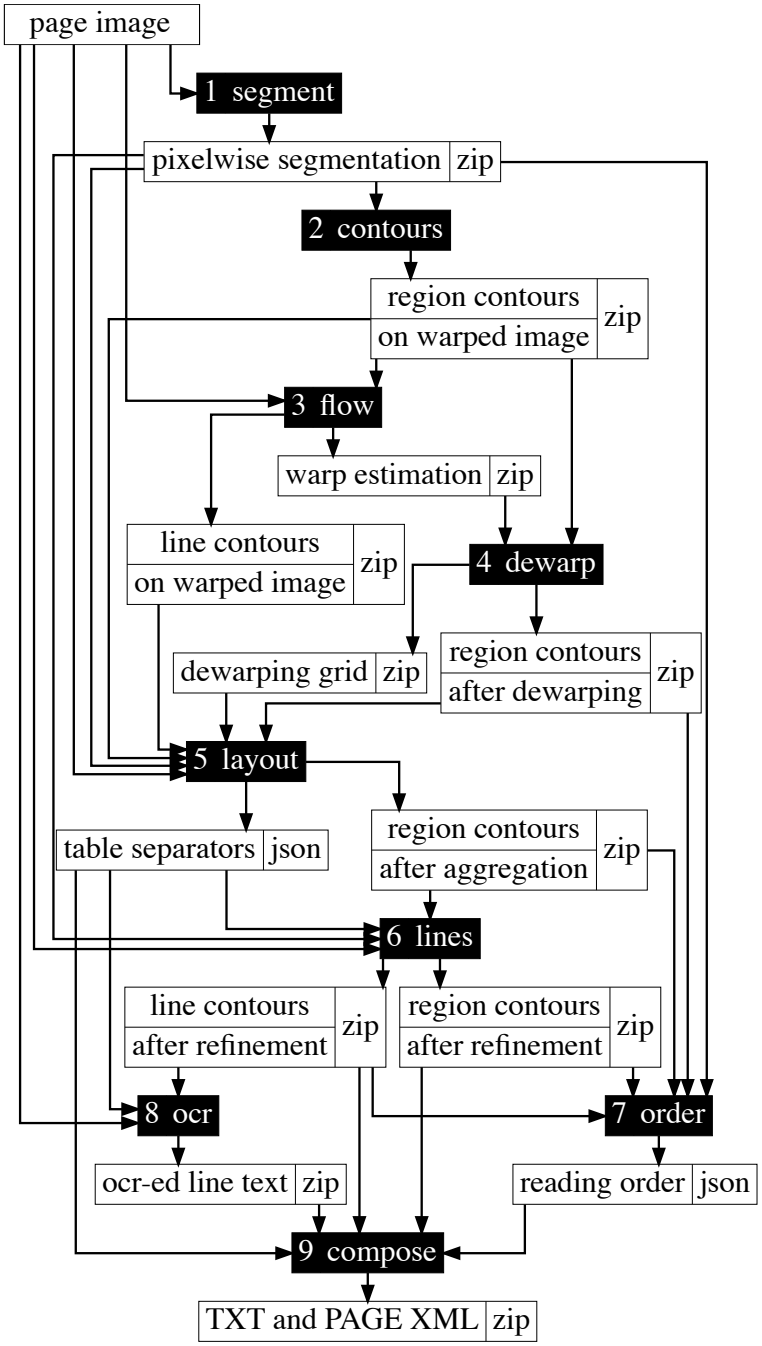


Figure 1: *Origami*'s internal data flow. Filled black boxes are stages triggered by user.

baselines from pixel-based DNN output. Implementations for the latter task can get complex, as for example the implementation of the OCR-D baseline detector[18] shows. Note also that we do not want to simply estimate the separators as straight lines by using principal component analysis (PCA) or Hough transforms [7], since separator lines might be curved back and forth

[18] https://github.com/qurator-spk/sbb_textline_detection

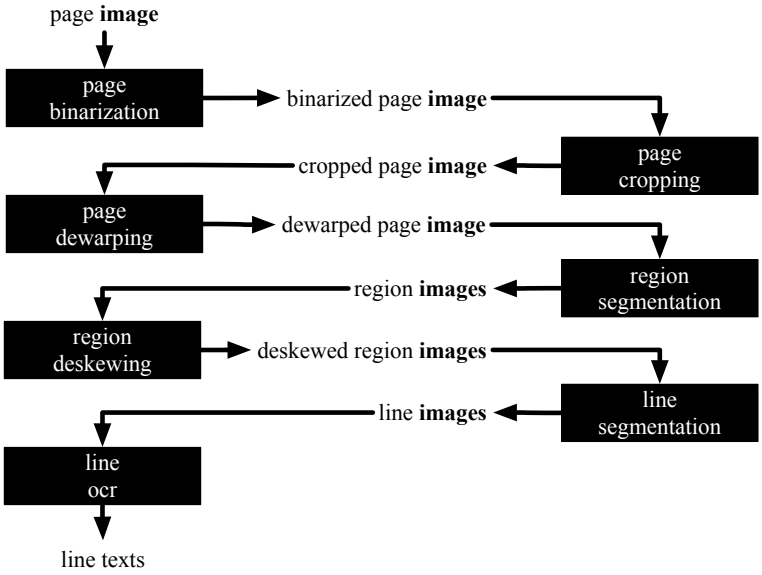


Figure 2: Typical OCR-D workflow (simplified).

and we will use that curvature later for warping estimation.

One approach that seems to work very well, is to find the contours and then compute a thinning transform like the medial axis or the straight skeleton [1]. The latter are well understood and common algorithms of computational geometry. Unfortunately, computing skeletons in traditional geometry libraries such as CGAL [10] is magnitudes too slow for our use case with highly detailed polygons[19]. We therefore implemented a much faster discrete (i.e. image-based) skeleton based on the algorithm in [15]. We extend this algorithm to extract detailed paths between detected nodes. Based on the known separator orientation, we then convert the extracted network structure to a directed acyclic graph in order to be able to efficiently find the longest path in terms of the euclidean distance [42]. Our implementation is rather simple (about 200 lines of code), fast (we use Numba [26] for just-in-time compilation) and robust. With this modification, our contours step runs in less than one second for a typical page.

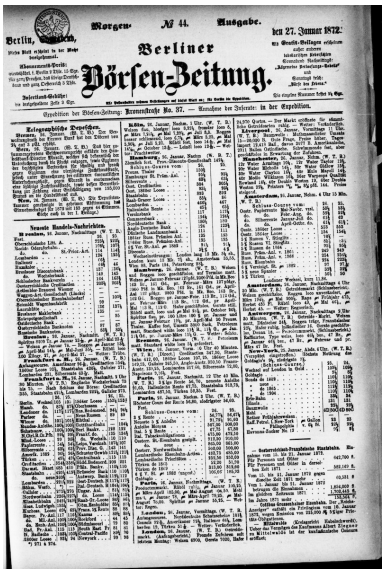
Combining this polyline extraction with a pixelwise baseline detection DNN in a two-stage process would yield a design similar to Grüning et al. [19] or the SBB textline detector[21]. This approach could replace Tesseract as baseline extraction technology in stages 3 and 6.

3.3. Flow and dewarp (stages 3 & 4)

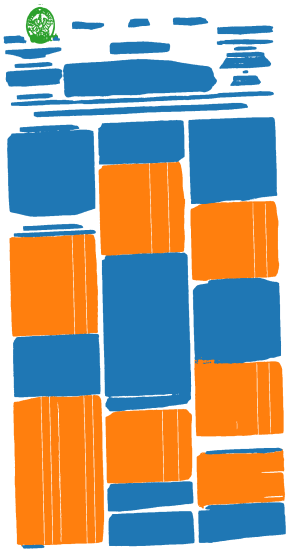
This section covers both stages 3 and 4 since they are conceptually interwoven. The reason we split them in two in our architecture, is that we want to emphasize that our design does not assume them as one monolithic stage, which is a common practice in many other systems. For example, it would be rather easy to replace stage 3 with a different implementation (e.g. a DNN-based warping detection) in the future.

There are numerous approaches to dewarping (for an overview, see Chapter 8 in [8] and

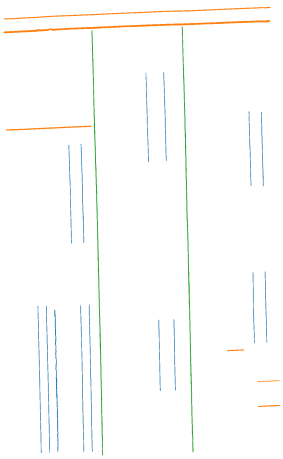
[19] We ported CGAL's skeleton functionality to *scikit-geometry*[20] for this project.
[21] https://github.com/qurator-spk/sbb_textline_detection



(a) Input document: Scan of the BBZ from January 27, 1872, *Morgenausgabe*, front page.



(b) Predictions for region labels *TEXT* (blue), *TABULAR* (orange) and *ILLUSTRATION* (green).



(c) Horizontal separators H (red), vertical text separators V (green) and table column separators T (blue).

Figure 3: A sample page from the Berliner Börsen-Zeitung (a) shown with the regions (b) and separator lines (c) automatically inferred from the scanned image by our DNN in stage 1.

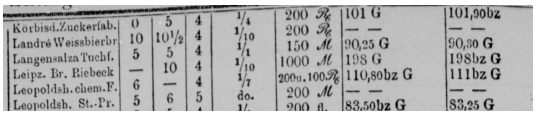
[17]), but in practice, very few implementations are actually available. A number of recent approaches use deep neural networks (DNNs). For example, OCR-D's official page dewarping module *ocrd-anybaseocr-dewarp*[22] is based on the *pix2pixHD* GAN [4, 9, 34]. DocUNet, a different recent DNN approach, is based on U-Nets [30]. While we found the *ocrd-anybaseocr-dewarp* module works quite well for low-resolution documents, we were not able to apply it for our use case as we received pixel clutter when running it on the full page resolution (we used binarized input as the module documentation states [34]). As shown in Figure 4, dewarping the original image (a) is achieved, but the output is not readable anymore (b). Only when we reduced the input to a low-resolution detail region of the page, we obtained readable output (c) (though still of somewhat worse quality than the binarized input).

For the digitization of newspapers, layout analysis and reading order analysis strongly benefit from a fully dewarped page, so the reported issues with *ocrd-anybaseocr-dewarp* disqualified it for being used in our pipeline. Another reason to abandon *ocrd-anybaseocr-dewarp*, is that it was trained to expect and generate binarized images, which would conflict with *Origami*'s non-binarized line OCR model[23]. As an alternative, we looked into dewarping with more traditional (i.e. non-DNN) approaches. However, there seem to be only two working open source implementations available: OCR-D's *ocrd-cis-ocropy-dewarp* and Leptonica [24]. Both employ rather simple warping detection heuristics that are not well suited for multi-column pages and

[22]The module originates from the DFKI's *anyOCR* package, see https://github.com/OCR-D/ocrd_anybaseocr.

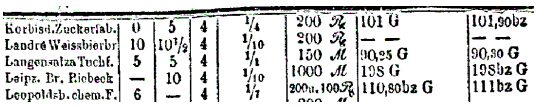
[23]We use *Origami* with a line OCR model that relies on non-binarized input, as we obtained lower character error rates than with binarized input in previous experiments. Also see Section 3.6

[24]http://www.leptonica.org/dewarping.html

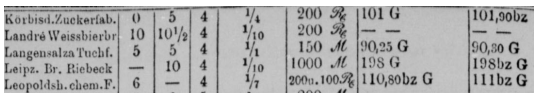


(a) Detail from page 10 from August 22, 1888.

(b) *Ocrd-anybaseocr-dewarp* run on full page (detail).



(c) *Ocrd-anybaseocr-dewarp* run on small region.



(d) *Origami* dewarping run on full page (detail).

Figure 4: Results of running various dewarping implementations on a warped sample page. Image (a) shows one region of interest in the warped original. Images (b) and (c) show the same region after applying *ocrd-anybaseocr-dewarp* on page and region level. Image (d) shows the detail after applying *Origami*'s dewarping to the full page.

complex layouts. Neither can therefore be expected to produce good output when applied to the region or page level [34]. To summarize, neither of the available implementations are suitable for dewarping high-resolution grayscale images with complex layouts, nor can they be extended or modularized to add this missing functionality. Consequently, we decided to implement a dewarping step for the page level from scratch that can take any form of warping information in the form of a number of points and flow lines on the page and generate a dewarped page - thereby separating the inner workings of the warping estimation (for example via some form of external baseline detection algorithm) from the actual dewarping calculation. In contrast to a neural network, the process is highly transparent from the algorithmic perspective and also scales to any resolution and color depth. A sample output of our approach can be seen in Figure 4 (d).

Our approach will be described in more detail in the following passage: After we first experimented with constrained cylinder models, such as in [53], we settled on the versatile vector field model of Schneider et al.[25] [41]. Instead of the baseline detection described by Schneider et al., we use Tesseract's baseline detection, which we run for each of the detected regions from our previous step. As we only present one homogeneous, single-column text region at a time, this works very well[26]. In addition to baselines, we extract direction information from the vectorized separator paths from the previous stage. Figure 5 shows an example of this process.

In a next step, all sample points (i.e. points and their derived flow directions) are used to build two vector fields that model page flow lines in horizontal and vertical direction respectively. As proposed in [41], we use a Delaunay triangulation to interpolate all missing values. Since we work with fewer samples (we generate one sample per line), we also need to extrapolate the outside of the convex hull of sample points (this does not seem to be necessary in [41]). This turns out to be an important but complex technical detail, necessary for good results with our current setup. In a final step, we build a combined dewarping grid that models discrete intersections of the flows of both vector fields (instead of processing them one

[25]Somewhat similar grid based approaches have been proposed by others, for example [39]

[26]Unfortunately, Tesseract's official API only provides straight baselines at the moment, and our approach (and the remaining warping in Figure 5) could be improved by either accessing Tesseract's internal spline baselines or using a different (e.g. DNN-based) baseline estimation altogether.

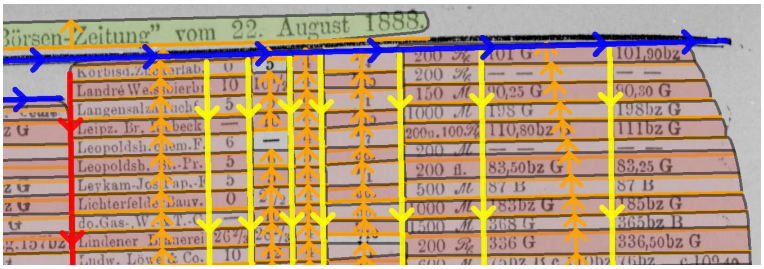
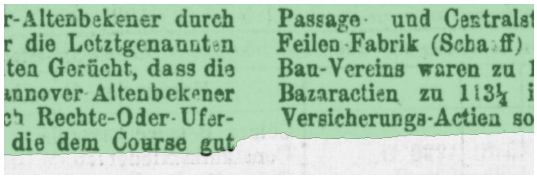


Figure 5: Sampling page warp from baselines and separator directions. Orange arrows show baselines' up vectors. Red, blue and yellow lines show various separator types and their extracted directions.



(a) Example of undersegmentation. These two columns should be separate regions, not one.

(b) Example of oversegmentation. The yellow table should consist of one region, not two.

Figure 6: Two examples of typical segmentation errors that the layout stage tries to fix with heuristic rules. Green lines in (b) are detected table separators T.

after another, as suggested in [41]). We also sample and modify grid borders to ensure that no content is dewarped outside the final page frame. The dewarping representation resulting from this step is stored as a grid of point locations.

3.4. Layout and lines (stages 5 & 6)

This rule-based stage attempts to fix typical segmentation errors (for an overview, see [2, 43]) by merging and splitting regions. As illustrated in Figure 6, some regions that got connected by the DNN in stage 1, should not be connected, while other regions that did not get connected, should actually get merged. This stage addresses these issues through a configurable pipeline of fast geometric operators. In terms of mathematical morphology, these operators can be likened to a series of highly selective dilations and erosions. The name "layout" for this stage is a bit of a misnomer, since the layout detection is actually spread over various stages (starting at stage 1) - yet this is the stage that finally commits to an overall region layout.

Although similar operations have been implemented in many segmentation solutions through binary morphological operations (like dilation and erosion) on a pixel level [2, 11], we are not aware of a description of this specific framework of selective polygonal operators for this use case. Details of the most useful operators are presented in the following.

Definitions We first define some useful symbols. Note that we use the terms shape and region interchangeably. $H(A)$ is a hull operator on a shape A as described in Section 3.4, $\Lambda(A)$ refers to the number of baselines detected in a region A, $\mathrm{d}(A, B)$ is the shortest distance between A and B, and $x_0(A)$ ($y_0(A)$) and $x_1(A)$ ($y_1(A)$) refer to the minimum and maximum x (y) coordinates of the region. We denote the area of region A as $|A|$ and the length of the interval $[u_0, u_1]$ as $|u|$. We refer to heuristic parameters as α or, for multiple parameters, as $\alpha_1, ..., \alpha_n$ (these

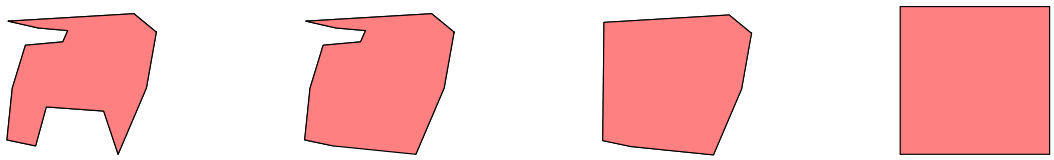


Figure 7: Hull operators supported in *Origami*: base form (a), concave hull (b), convex hull (c), and rectangular form (d).

parameters are always local to the operator described). All configured parameters relating to lengths or areas on the page are handled independently of resolution by scaling them according to the page size. Finally, we define a measure of overlap $\mathrm{OVL}(A,B) = \frac{|A \cap B|}{min(\downarrow|A|,\downarrow|B|)}$ for shapes or intervals A, B and a measure of cohesion $\mathrm{COH}(S) = \frac{\sum_x^S |x|}{|H(\cup S)|}$ for some set of regions S.

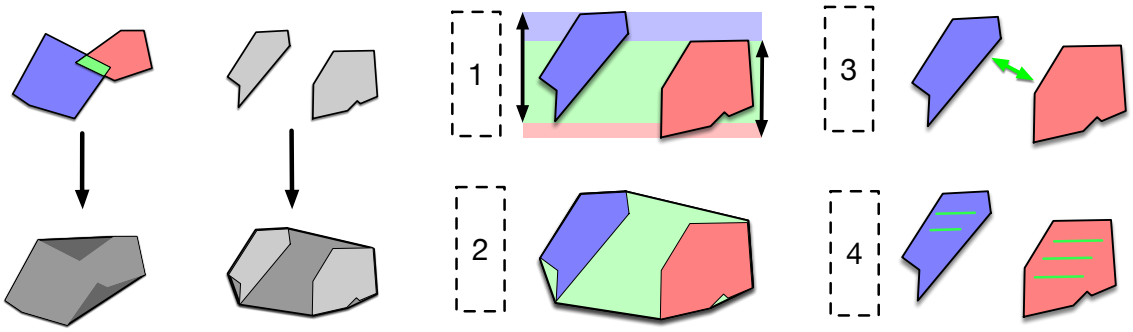
The hull operator This operator extends a geometric region to some form of hull that is defined by additional parameters. It never grows the region beyond its axis-aligned bounding box (AABB). As shown in Figure 7, *Origami* supports expanding a shape to a concave hull (the latter is defined by two *concavity* parameters, see [36]), the convex hull, or a rectangular form (the AABB).

The rationale for this step is to find a better approximation of an area that might only have been partially detected by the DNN. For example, if we know we are dealing with a Manhattan layout, we might benefit from finding overlaps in rectangular shapes. Note that we also apply these hulls when merging shapes in the other operators. For the BBZ, we use convex hull operators.

The overlap merge operator This operator merges two regions A, B of the same region type, e.g. *TEXT*, if $\mathrm{OVL}(A,B) >= \alpha$, i.e. if the green area in the leftmost example in Figure 8(a) gets too big in comparison to the unmerged regions. The merged region is given as $H(A \cup B)$.

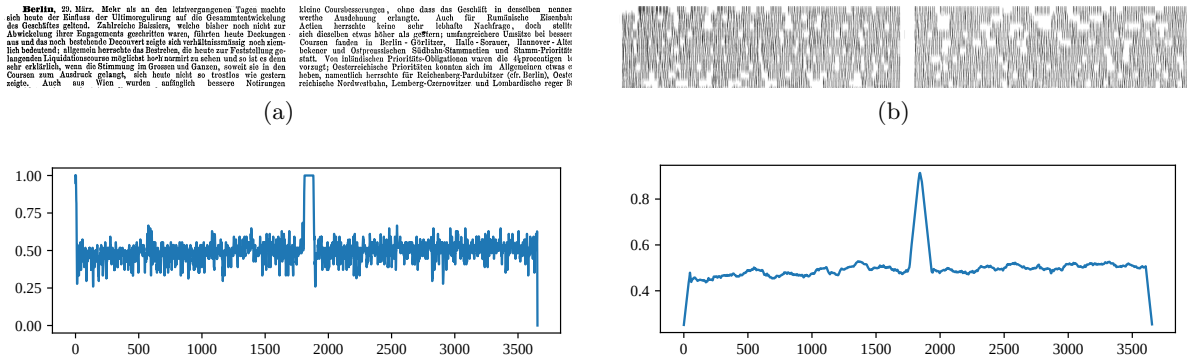
The adjacency merge operator This operator merges regions that have the same type (e.g. *TEXT*) that are near to each other, subject to additional constraints. We use this to extend regions that are vertically aligned to the left or right (e.g. for finding header texts), and regions that are horizontally aligned to the bottom or right (e.g. for merging regions of the same table). Specifically, we merge A and B if (1) the shapes fulfill a specified horizontal (or vertical) alignment, e.g. $\mathrm{OVL}([x_0(A), x_1(A)], [x_0(B), x_1(B)]) > \alpha_1$, (2) $\mathrm{COH}(A \cup B) > \alpha_2$, (3) $d(A,B) < \alpha_3$, and (4) $max(\Lambda(A), \Lambda(B)) <= \alpha_4$ (see Figure 8). Additionally, we check if a newly merged region overlaps with other region types or with separators. If any of the two overlaps are above a certain threshold, we do not merge. To determine adjacency efficiently, we build a segment-based Voronoi-diagram from the region contours.

The sequential merge operator Sequential merge is similar to adjacency merge, but compares longer runs of regions along a reading order determined by a X-Y cut on the regions (see Section 3.5). This has proven useful to merge cluttered table regions as shown in Figure 6(b). We greedily look for runs of regions S with $\mathrm{COH}(S) > \alpha_1$ and only stop extending a run if we



(a) Two types of merges. (b) The four criteria of adjacency merge.

Figure 8: (a) Overlap merge (left) and adjacency merge (right), when H (the hull operator) is configured to form the convex hull (the additional area added by applying H is indicated in darker gray in the merged forms at the bottom). (b) Conditions for merging adjacent regions: (1) vertical alignment (indicated as green area), (2) cohesion, a measure of the excess area we would add when actually merging the regions (indicated in green), (3) shortest distance between regions (green arrow), (4) number of text lines in the involved regions.



(a) (b) (c) (d)

Figure 9: Steps involved in spillover detection: (a) binarization, (b) vertical convolution, (c) quantiles, (d) horizontal convolution. (c) and (d) show horizontal pixel offset on the x-axis and grayscale value on the y-axis (1 is white, 0 is black).

encounter $\mathrm{COH}(S) < \alpha_2$, in which case we start a new run. Similar to adjacency merge, we only merge if $d(A, B) < \alpha_3$ and no overlaps with other region types or separators exist.

The fix-spillover operator This operator fixes undersegmentation, i.e. cases where the DNN recognized one region, whereas it should have recognized separate regions as shown in Figure 6(a). It is implemented as a morphological analysis followed by a split. As shown in Figure 9, we first binarize a region using a Sauvola threshold (a), then convolve vertically with a kernel that matches the line height (b). For each column, we now find the 0.1 quantile (c). Finally, we convolve horizontally (d). In this signal, we find peaks of a certain minimum width, which are the desired whitespace columns. The region is then split at this location. The whole operation depends on proper deskewing and dewarping.

Figure 10: Example of our two-stage X-Y cut approach: The numbers in circles indicate the determined reading order. In this example, 12-13-14-15 originally belonged to one region and 16 to another region. On the region level, these regions' bounding boxes would intersect, and a clean X-Y cut is not feasible. We detect this case, split these two regions into their lines, and re-run an X-Y cut on the line level (split lines are underlined in orange).

Lines stage After the layout completes, the new determined regions are used to extract new baselines. This task is much easier than before (i.e. in stage 3), since dewarping will have made lines (roughly) horizontal and parallel. This step currently uses Tesseract internally.

3.5. Reading order (stage 7)

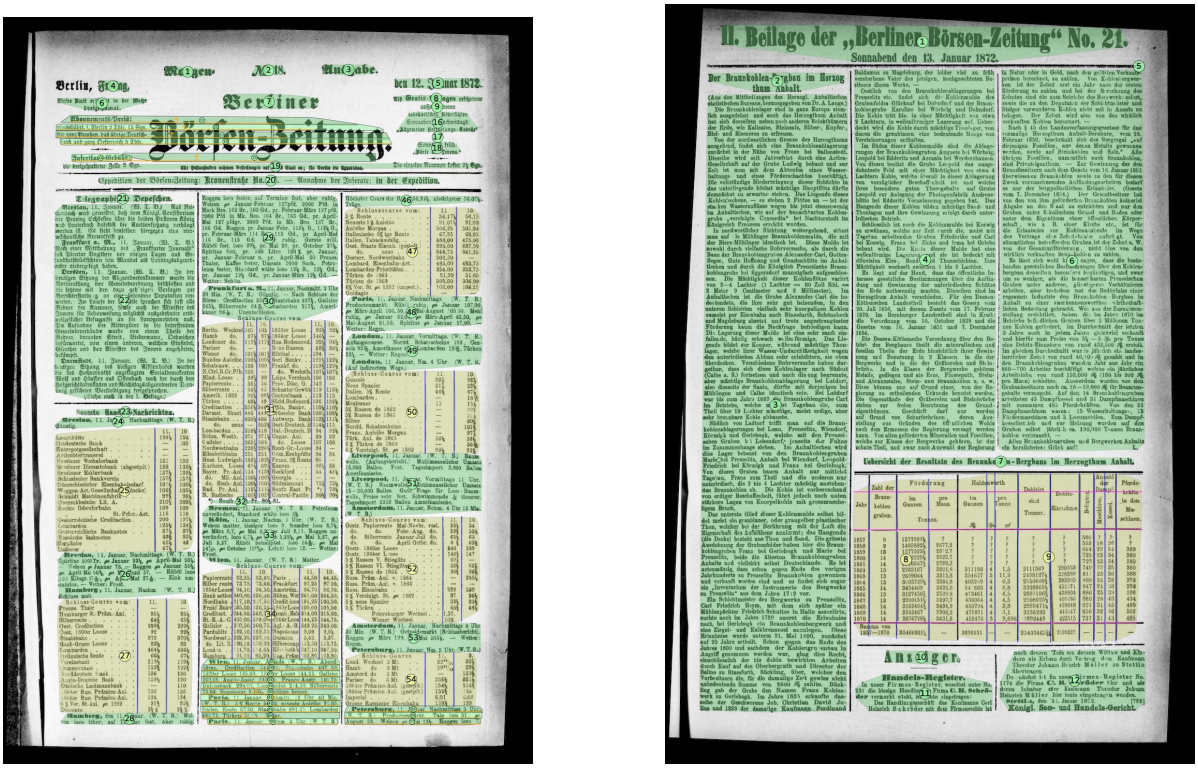
Reading order is determined using a recursive X-Y cut (also known as *RXYC*) [32, 11, 31] on the dewarped regions[27]. We project bounding boxes as described in [20] and use a custom scoring scheme (similarly non-standard schemes have been investigated by Meunier [31]). *Origami* supports three variants when choosing a cut: (1) widest gap, (2) longest cut, (3) largest whitespace area (the product of (1) and (2)). We found that (3) works best for our corpus. We combine this score with a separator scoring scheme that prefers cuts that run along aligned separators, while it penalizes cuts that cross unaligned separators (e.g. we will penalize a vertical cut that would split a horizontal separator). For this, we measure the ratio of the whitespace gap that is covered by a separator.

As is illustrated in Figure 10, we employ a two-step approach X-Y cut that refines regions into lines for those regions whose bounding boxes overlap with each other (and do therefore not provide whitespace gaps for proper cuts). This approach can fix some simple border cases of notoriously difficult L-shapes [31].

Figure 11 provides some examples for the consolidated information about layout regions, reading order, and table structure (i.e. table regions and their columns) at this stage of the workflow.

Before X-Y cuts are performed on the region level, a refinement step resamples evidence from the DNN output from stage 1 at the line level: for each line in a region, we check if the majority of the DNN pixel predictions for the line's area corresponds to the region type (e.g. *TEXT*). If this is not the case, the line gets deleted (and the region is reduced accordingly). This step resolves overlaps at some region boundaries and thus provides a more robust basis

[27] Proper dewarping is crucial here, since for X-Y cut to work, whitespace areas and separator lines need to be straight and axis-aligned.



(a) (b)

Figure 11: Examples of completed layout and reading order analysis (numbers in circles indicate reading order). The black borders stem from page dewarping.

for X-Y cuts.

3.6. OCR and compose (stages 8 & 9)

We use an OCR model specifically trained for the BBZ with Calamari as well as a custom network architecture and various data augmentation techniques. The model uses ensemble voting with 5 folds and non-binarized line images as input. We refer to this model as *Origami*'s BBZ model, as *Origami* supports specifying other (and binarizing) OCR models as well. We described the training of *Origami*'s BBZ model in detail in a separate paper [29].

The following results should be understood as a measure of what can be achieved with *Origami* under optimal conditions, i.e. when using a tailored OCR model for a specific corpus. While we believe that the results illustrate the pipeline's potential, our results do not provide a general baseline when applying *Origami* to other corpora or in terms of other pipeline's general performance. To really assess the quality of *Origami* in comparison to other systems such as OCR-D or Transkribus, one would need to define a mapping between similar sets of stages and then test against a shared set of inputs and validation outputs. Defining such a methodology is a very challenging task though, which we may investigate in more detail in future work[28].

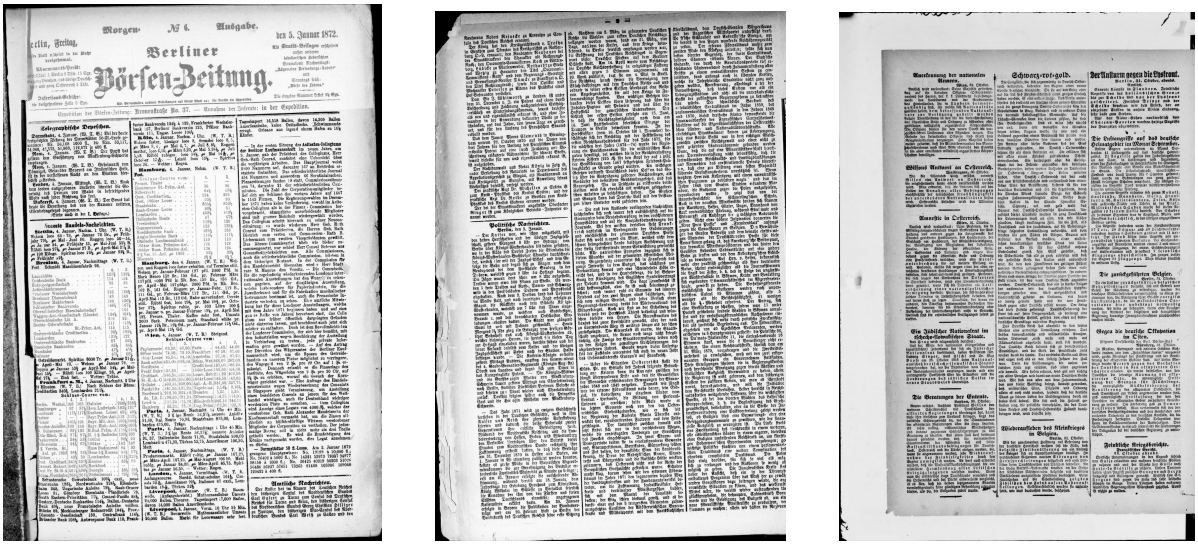
Table 2 lists the results on three typical BBZ test pages, which are shown in Figure 12. As indicated by the light areas in Figure 12(a), we excluded a badly readable header area (which

[28] For a discussion of the problems involved, see [38, 13, 23].

Table 2
CERs and WERs for the final stage without table cell text (best scores are bold). All models used the same line image input from previous *Origami* stages. Non-*Origami* models' input was always binarized using an Otsu threshold. Comparisons were calculated after harmonizing obviously different transcription schemes (e.g. dashes and Fraktur-s). (a) contains mixed Fraktur and Antiqua content, which explains the bad scores for the Fraktur-only models. The bad scores for (c) for non-*Origami* models seem to be a result from (c)'s lower line height (this hypothesis is supported by the fact that larger title headings come out correctly most of the time, whereas the smaller body text is mostly garbage).

	CER			WER		
model	(a)	(b)	(c)	(a)	(b)	(c)
OCR-D gt4histocr-calamari[1]	16.37%	2.98%	18.98%	53.18%	15.18%	60.4%
Calamari fraktur_19th_century[2] 2019[3]	28.83%	4.7%	36.98%	70.41%	22.13%	84.87%
Calamari fraktur_19th_century[2] 2020	14.23%	1.37%	20.3%	42.23%	5.99%	60.02%
Origami, trained on BBZ (v19-h2-e-v1)	**0.76%**	**0.3%**	**0.32%**	**2.4%**	**1.39%**	**1.75%**

[1] Pretrained models from https://github.com/OCR-D/ocrd_calamari
[2] Pretrained models from https://github.com/Calamari-OCR/calamari_models
[3] Last year's pretrained model, i.e. git commit hash e568b1d



(a) (b) (c)

Figure 12: Page scans used to measure overall OCR performance: (a) January 5, 1872, *Morgenausgabe*, header page, 4582×6507 pixels, overall bad printing quality, (b) January 5, 1872, *Morgenausgabe*, page 2, 4301×6521 pixels, (c) November 1, 1918, *Morgenausgabe*, page 2, 2400×3555 pixels. Light areas in (a) were excluded from evaluation. (a) contains both Fraktur and Antiqua, whereas (b) and (c) contain only Fraktur. Note that (a) and (b) are high resolution images (body text line height about 50 pixels), whereas (c) is a lower resolution image (body text line height about 25 pixels).

also contained various recurring phrases trained through other pages' header areas, that would have given *Origami* an unfair advantage) and several table regions with low quality text (and fractions) from the evaluation of page (a) for all models. No parts of the evaluation pages were included in *Origami*'s training set.

Origami's BBZ model outperforms the best other currently available Fraktur models by at

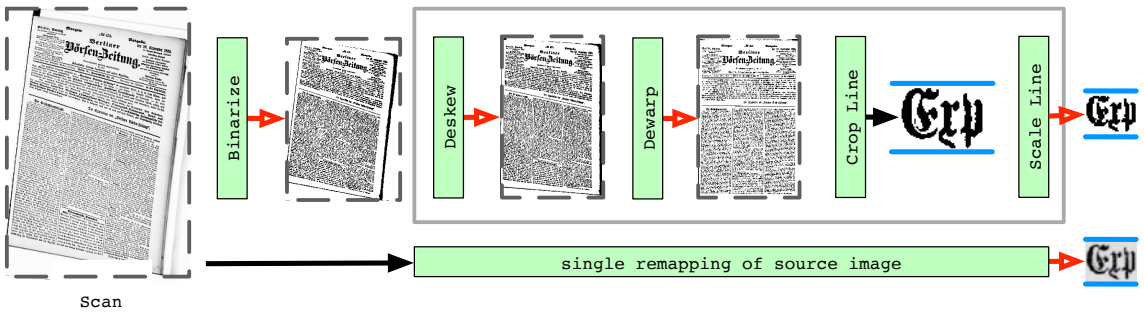


Figure 13: Line image sampling implemented in *Origami*. Conventional OCR pipelines (top path) perform deskewing, dewarping, scaling and other preprocessing operations (e.g. binarization) in subsequent isolated steps, thereby risking degrading quality with each step (red arrows indicate possible information loss due to sampling, crop does not involve sampling). *Origami* (bottom path) combines deskewing, dewarping and scaling into one single remapping (a set of piecewise linear mappings built through corresponding control points [18]), thus the original source image is only sampled once.

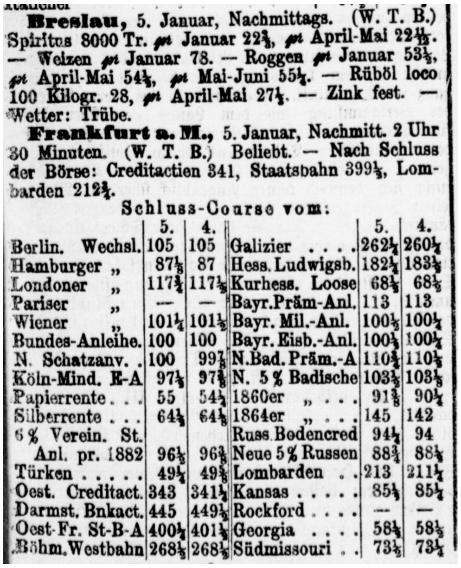
least 1% for CER (character error rate) and by at least 5% for WER (word error rate)[29]. In general, the gap seems to be larger. The absolute WERs of around 2% obtained in this small evaluation compares quite favorably to the WERs of above 5% recently reported for ABBY 10 and 11 when run on newspapers published between 1884 and 1947 [24]. Our results for page (a) illustrate the high error rates one obtains when applying pretrained Fraktur-only models to mixed Antiqua-Fraktur content. Page (b) shows the performance under ideal conditions (high image resolution, only Fraktur text) with pretrained third-party models. With page (c), third party models' quality plummets with lower image resolutions, whereas *Origami*'s own model roughly retains its performance. This might make *Origami*'s BBZ model an option for working with lower resolution input data. *Origami*'s BBZ model understands both Fraktur and Antiqua at similar error levels. It was also trained to recognize numerical fractions and a number of special symbols found in the BBZ. Finally, it was trained to correctly handle increased letter-spacing (*Sperrsatz*) that is commonly found in the BBZ.

To minimize lost information through resampling, *Origami* combines deskewing, dewarping and scaling into a single linear remapping from the source image to the final line image, that is directly fed into the line-level OCR network (see Figure 13). Many pipelines perform these operations step by step using subsequent (low-quality nearest-neighbor) resampling on already binarized input (see top half of Figure 13), which seems not ideal given that resampling is a lossy process [16]. Any output of this stage is finally composed into a plain text version and a schema-validated *PAGE XML* file (the latter also contains polygonal coordinates for all regions and the obtained reading order). Figure 14 shows an example of the plain text output that features both body text and tables.

4. Conclusion

In this paper we presented *Origami*, a new end-to-end pipeline for performing OCR on historical newspapers. *Origami* is robust enough for large scale batches in production environments, but also small and flexible enough to allow for easy experimentation with new algorithmic

[29] CERs and WERs were evaluated using Dinglehopper (https://github.com/qurator-spk/dinglehopper).



Breslau, 5. Januar, Nachmittags. (W. T. B.) Spiritus 8000 Tr. pr Januar 22⅜, pr April-Mai 22 11/12. — Weizen pr Januar 78. — Roggen pr Januar 53½, pr April-Mai 54½, pr Mai-Juni 55¼. — Rüböl loco 100 Kilogr. 28, pr April-Mai 27½. — Zink fest. — Wetter: Trübe.

Frankfurt a. M., 5. Januar, Nachmitt. 2 Uhr 30 Minuten. (W. T. B.) Beliebt. — Nach Schluss der Börse: Creditactien 341, Staatsbahn 399½, Lombarden 212¾.

Schluss-Course vom:

	5.	4.		5.	4.
Berlin. Wechsl.	105	105	Galizier	262¼	260¼
Hamburger „	87⅛	87	Hess. Ludwigsb.	182¼	183⅛
Londoner „	117¾	117½	Kurhess. Loose	68½	68⅛
Pariser „	—	—	Bayr.Präm-Anl.	113	113
Wiener „	101¼	101⅛	Bayr. Mil.-Anl.	100⅛	100¼
Bundes-Anleihe.	100	100	Bayr. Eisb.-Anl.	100⅜	100¼
N. Schatzanv. .	100	99⅞	N.Bad. Präm.-A	110¼	110⅛
Köln-Mind. E-A	97½	97⅜	N. 5 % Badische	103⅜	103⅛
Papierrente . . .	55	54½	1860er „ . . .	91⅜	90¼
Silberrente . . .	64⅛	64⅜	1864er „ . . .	145	142
6 % Verein. St.			Russ.Bodencred	94¼	94
Anl. pr. 1882	96½	96⅜	Neue 5 % Russen	88¾	88⅛
Türken	49¼	49⅝	Lombarden . .	213	211¼
Oest. Creditact.	343	341½	Kansas	85⅛	85¼
Darmst. Bnkact.	445	449½	Rockford . . .	—	—
Oest-Fr. St-B-A	400½	401⅛	Georgia	58⅛	58⅛
Böhm.Westbahn	268½	268⅛	Südmissouri . .	73⅛	73¼

(a) Scan (detail).

```
Breslau, 5. Januar, Nachmittags. (W. T. B.)
Spiritus 8000 Tr. pr. Januar 22<3/3>, pr. April-Mai 22<11/12>.
-- Weizen pr. Januar 78. -- Roggen pr. Januar 53<1/2>,
pr. April-Mai 54<1/2>, pr. Mai-Juni 55<1/4>. -- Rüböl loco
100 Kilogr. 28, pr. April-Mai 27<1/2>. -- Zink fest. --
Wetter: Trübe.
Frankfurt a. M., 5. Januar, Nachmitt. 2 Uhr
30 Minuten. (W. T. B.) Beliebt. -- Nach Schluss
der Börse: Creditactien 341, Staatsbahn 399<1/2>, Lom-
barden 212<3/4>.
Schluss-Course vom:

+------------------+----------+----------+------------------+----------+----------+
|                  | 5.       | 4.       |                  | 5.       | 4.       |
| Berlin. Wechsl.  | 105      | 105      | Galizier....     | 262<1/4> | 260<1/4> |
| Hamburger ,,     | 87<1/8>  | 87       | Hess. Ludwigsb.  | 182<1/4> | 183<1/2> |
| Londoner ,,      | 117<3/4> | 117<1/2> | Kurhess. Loose   | 68<1/2>  | 68<1/2>  |
| Pariser ,,       | --       | --       | Bayr. Präm-Anl.  | 113      | 113      |
| wiener ,,        | 101<1/4> | 101<1/2> | Bayr. Mil.-Anl.  | 100<1/2> | 100<1/4> |
| Bundes-Anleihe.  | 100      | 100      | Bayr. Eisb.-Anl. | 100<3/2> | 100<1/4> |
| N. Schatzanv..   | 100      | 99<7/8>  | N. Bad. Präm.-A  | 110<3/4> | 110<1/2> |
| Köln-Mind. E-A   | 97<1/2>  | 97<8>    | N. 5% Badische   | 103<1/2> | 103<1/8> |
| Papierrente...   | 55       | 54<1/4>  | 1860er ,,...     | 91<3/8>  | 90<1/4>  |
| Silberrente...   | 64<1/2>  | 64<1/8>  | 1864er ,,...     | 145      | 142      |
| 6% Verein. St.   |          |          | Russ Bodencred   | 94<1/4>  | 94       |
| Anl. pr. 1882    | 96<1/2>  | 96<3/8>  | Neue 5% Russen   | 88<3/4>  | 88<1/2>  |
| Türken.....      | 49<1/4>  | 49<5/8>  | Lombarden..      | 213      | 211<1/4> |
| Oest. Creditact. | 343      | 341<1/4> | Kansas.....      | 85<1/2>  | 85<1/4>  |
| Darmst. Bnkact.  | 445      | 449<1/2> | Rockford....     | --       | --       |
| Oest-Fr. St-B-A  | 400<1/4> | 401<1/2> | Georgia....      | 58<1/2>  | 58<1/2>  |
| Böhm. Westbahn   | 268</2>  | 268<2>   | Südmissouri..    | 73<1/2>  | 73<1/4>  |
+------------------+----------+----------+------------------+----------+----------+
```

(b) Result of compose stage.

Figure 14: Detail of a scan of the BBZ from January 6, 1872, *Morgenausgabe*, page 1 (a) and the corresponding OCR result from the plain text compose stage (b).

approaches. *Origami* offers a reliable, table-aware layout detection based on deep neural networks, high-resolution deskewing and dewarping, selective polygonal merge and split operators and separator-aware, two-stage X-Y cuts. We also demonstrated (1) the feasibility of an all-in-one transformation for extracting scaled, dewarped line images directly from the original document scan without intermediary sampling, and (2) the validity of a new non-standard paradigm regarding binarization that only relies on thresholding as a source of information for micro decisions (e.g. layout operators), but not as general pre-processing for all subsequent stages. Our evaluations showed how the pipeline's mixed Fraktur-Antiqua OCR model with non-binarized input (trained with Calamari) clearly outperforms other publicly available single-typeface ensemble models in terms of error rates for our use case.

In our current project on digitizing the Berliner Börsen-Zeitung, the *Origami* pipeline proved to be the ideal experimental platform to research how existing high-quality OCR components can be leveraged and how new functionalities can be implemented. As *Origami* is lean, flexible and generally easy-to-use (all components are bundled and can be setup with just a few commands), we hope it will be useful for others as well. We also hope that the *Origami* workflow will spark some more discussion on some of the experimental features we implemented, for instance the general role and necessity of binarization as a pre-processing step.

Acknowledgments

This research was supported by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation), project number BU 3502/1-1.

References

[1] O. Aichholzer et al. “A Novel Type of Skeleton for Polygons”. In: *J. UCS The Journal of Universal Computer Science* 1 (Jan. 1995), pp. 752–761. DOI: 10.1007/978-3-642-80350-5_65.

[2] A. Antonacopoulos et al. “ICDAR 2013 Competition on Historical Newspaper Layout Analysis (HNLA 2013)”. In: *Proceedings of the 12th International Conference on Document Analysis and Recognition (ICDAR 2013)*. Washington, DC, USA, 2013, pp. 1454–1458.

[3] K. Baierer et al. “OCR-D kompakt: Ergebnisse und Stand der Forschung in der Förderinitiative”. In: *BIBLIOTHEK – Forschung und Praxis* (June 2020). DOI: 10.18452/21548.

[4] V. K. Bajjer Ramanna, S. S. Bukhari, and A. Dengel. “Document Image Dewarping Using Deep Learning”. In: *Proceedings Ot the International Conference on Pattern Recognition Applications and Methods (ICPRAM 2019)*. Prague, Czech Republic: Insticc, Feb. 2019, pp. 524–531. DOI: 10.5220/0007368405240531.

[5] R. Barman et al. “Combining Visual and Textual Features for Semantic Segmentation of Historical Newspapers”. In: *CoRR* abs/2002.06144 (May 2020). arXiv: 2002.06144 [cs].

[6] S. Bianco et al. “Benchmark Analysis of Representative Deep Neural Network Architectures”. In: *IEEE Access* 6 (2018), pp. 64270–64277. ISSN: 2169-3536. DOI: 10.1109/ACCESS.2018.2877890. arXiv: 1810.00736.

[7] G. Bradski. “The OpenCV Library”. In: *Dr. Dobb’s Journal of Software Tools* (2000).

[8] S. S. Bukhari. “Generic Methods for Document Layout Analysis and Preprocessing”. Doctoralthesis. Technische Universität Kaiserslautern, 2012, p. 219.

[9] S. S. Bukhari et al. “anyOCR: An Open-Source OCR System for Historical Archives”. In: *Proceedings of the 14th International Conference on Document Analysis and Recognition (ICDAR 2017)*. Vol. 1. Kyoto, Japan, 2017, pp. 305–310.

[10] F. Cacciola. *2D Straight Skeleton and Polygon Offsetting*. Tech. rep. CGAL Editorial Board, 2020.

[11] R. Cattoni et al. *Geometric Layout Analysis Techniques for Document Image Understanding: A Review*. Tech. rep. Via Sommarive 18, I-38050 Povo, Trento, Italy: ITC-irst, 1998, pp. 1–68.

[12] C. Clausner, S. Pletschacher, and A. Antonacopoulos. “Aletheia - an Advanced Document Layout and Text Ground-Truthing System for Production Environments”. In: *Proceedings of the 11th International Conference on Document Analysis and Recognition (ICDAR 2011)*. Beijing, China: IEEE Computer Society, 2011, pp. 48–52. ISBN: 978-0-7695-4520-2. DOI: 10.1109/ICDAR.2011.19.

[13] C. Clausner, S. Pletschacher, and A. Antonacopoulos. “Flexible Character Accuracy Measure for Reading-Order-Independent Evaluation”. In: *Pattern Recognition Letters* 131 (Mar. 2020), pp. 390–397. ISSN: 01678655. DOI: 10.1016/j.patrec.2020.02.003.

[14] D. Dannélls, T. Johansson, and L. Björk. “Evaluation and Refinement of an Enhanced OCR Process for Mass Digitisation.” In: *Proceedings of the Digital Humanities in the Nordic Countries 4th Conference (DHN 2019)*. Ed. by C. Navarretta, M. Agirrezabal, and B. Maegaard. Copenhagen, Denmark, Mar. 2019, pp. 112–123. URL: http://www.ceur-ws.org/Vol-2364/9_paper.pdf.

[15] M. Dirnberger, A. Neumann, and T. Kehl. “NEFI: Network Extraction from Images”. In: *CoRR* abs/1502.05241 (2015). arXiv: 1502.05241 [cs].

[16] N. A. Dodgson. *Image Resampling*. Tech. rep. UCAM-CL-TR-261. University of Cambridge, Computer Laboratory, Aug. 1992. URL: https://www.cl.cam.ac.uk/techreports/UCAM-CL-TR-261.html.

[17] C. A. Glasbey and K. V. Mardia. “A Review of Image-Warping Methods”. In: *Journal of Applied Statistics* 25.2 (1998), pp. 155–171. DOI: 10.1080/02664769823151.

[18] A. Goshtasby. “Piecewise Linear Mapping Functions for Image Registration”. In: *Pattern Recognition* 19.6 (1986), pp. 459–466. ISSN: 0031-3203. DOI: 10.1016/0031-3203(86)90044-0.

[19] T. Grüning et al. “A Two-Stage Method for Text Line Detection in Historical Documents”. In: *International Journal on Document Analysis and Recognition (IJDAR)* 22.3 (Sept. 2019), pp. 285–302. ISSN: 1433-2833, 1433-2825. DOI: 10.1007/s10032-019-00332-1.

[20] J. Ha, R. M. Haralick, and I. T. Phillips. “Recursive X-Y Cut Using Bounding Boxes of Connected Components”. In: *Proceedings of 3rd International Conference on Document Analysis and Recognition (ICDAR 1995)*. Vol. 2. ICDAR 1995. Montreal, Quebec, Canada, 1995, pp. 952–955.

[21] M. J. Hill and S. Hengchen. “Quantifying the Impact of Dirty OCR on Historical Text Analysis: Eighteenth Century Collections Online as a Case Study”. In: *Digital Scholarship in the Humanities* 34.4 (Apr. 2019), pp. 825–843. ISSN: 2055-7671. DOI: 10.1093/llc/fqz024.

[22] I. Jerele et al. “Optical Character Recognition of Historical Texts: End-User Focused Research for Slovenian Books and Newspapers from the 18th and 19th Century”. In: *Review of the National Center for Digitization* 21 (2012), pp. 117–126. ISSN: 1820-0109.

[23] R. Karpinski, D. Lohani, and A. Belaïd. “Metrics for Complete Evaluation of OCR Performance”. In: *Proceedings of the International Conference on Image Processing, Computer Vision, & Pattern Recognition (IPCV 2018)*. Las Vegas, Nevada, USA, July 2018, pp. 23–29.

[24] L. Wilms. *Newspaper OCR Quality: What Have We Learned?* https://lab.kb.nl/about-us/blog/newspaper-ocr-quality-what-have-we-learned. July 2020.

[25] E. Klijn. “The Current State-of-Art in Newspaper Digitization”. In: *D-Lib Magazine* 14.1/2 (Jan. 2008). ISSN: 1082-9873.

[26] S. K. Lam, A. Pitrou, and S. Seibert. “Numba: A LLVM-Based Python JIT Compiler”. In: *Proceedings of the Second Workshop on the LLVM Compiler Infrastructure in HPC*. LLVM 2015. New York, NY, USA: Association for Computing Machinery, 2015, pp. 1–6. ISBN: 978-1-4503-4005-2. DOI: 10.1145/2833157.2833162.

[27] M. Li et al. “TableBank: Table Benchmark for Image-Based Table Detection and Recognition”. In: *CoRR* abs/1903.01949 (2019). arXiv: 1903.01949 [cs].

[28] B. Liebl and M. Burghardt. “An Evaluation of DNN Architectures for Page Segmentation of Historical Newspapers”. In: *CoRR* abs/2004.07317 (2020). arXiv: 2004.07317 [cs].

[29] B. Liebl and M. Burghardt. “On the Accuracy of CRNNs for Line-Based OCR: A Multi-Parameter Evaluation”. In: *CoRR* abs/2008.02777 (2020). arXiv: 2008.02777 [cs].

[30] K. Ma et al. “DocUNet: Document Image Unwarping via a Stacked u-Net”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR 2018)*. Salt Lake City, UT, USA, June 2018, pp. 4700–4709.

[31] J.-L. Meunier. “Optimized XY-Cut for Determining a Page Reading Order”. In: *Proceedings of the 8th International Conference on Document Analysis and Recognition (ICDAR 2005)*. Vol. 1. Seoul, South Korea, 2005, pp. 347–351.

[32] G. Nagy and S. C. Seth. “Hierarchical Image Representation with Application to Optically Scanned Documents”. In: *International Conference on Pattern Recognition*. 1984, pp. 347–349.

[33] C. Neudecker et al. “OCR-D: An end-to-end open source OCR framework for historical printed documents”. In: *Proceedings of the 3rd International Conference on Digital Access to Textual Cultural Heritage (DATeCH2019)*. New York, NY, USA: Association for Computing Machinery, May 2019, pp. 53–58. ISBN: 978-1-4503-7194-0. DOI: 10.1145/3322905.3322917.

[34] E. Engl et al. *OCR-D Workflows*. https://github.com/OCR-D/ocrd-website/blob/master/site/en/workflows.md. 2020.

[35] S. A. Oliveira, B. Seguin, and F. Kaplan. “dhSegment: A Generic Deep-Learning Approach for Document Segmentation”. In: *Proceedings of the 16th International Conference on Frontiers in Handwriting Recognition (ICFHR 2018)*. Niagara Falls, NY, USA: IEEE Computer Society, Aug. 2018, pp. 7–12. ISBN: ISBN 978-1-5386-5875-8. DOI: 10.1109/ICFHR-2018.2018.00011. arXiv: 1804.10371.

[36] J.-S. Park and S.-J. Oh. “A New Concave Hull Algorithm and Concaveness Measure for N-Dimensional Datasets”. In: *Journal of Information Science and Engineering* 29 (Mar. 2013), pp. 379–392.

[37] S. Pletschacher and A. Antonacopoulos. “The PAGE (Page Analysis and Ground-Truth Elements) Format Framework”. In: *Proceedings of the 20th International Conference on Pattern Recognition (ICPR 2010)*. Istanbul, Turkey, Aug. 2010, pp. 257–260.

[38] S. Pletschacher, C. Clausner, and A. Antonacopoulos. “Europeana Newspapers OCR Workflow Evaluation”. In: *Proceedings of the 3rd International Workshop on Historical Document Imaging and Processing (HIP 2015)*. Gammarth, Tunisia: ACM Press, 2015, pp. 39–46. ISBN: 978-1-4503-3602-4. DOI: 10.1145/2809544.2809554.

[39] M. Rahnemoonfar and A. Antonacopoulos. “Restoration of Arbitrarily Warped Historical Document Images Using Flow Lines”. In: *Proceedings of the 11th International Conference on Document Analysis and Recognition (ICDAR 2011)*. Beijing, China, Sept. 2011, pp. 905–909.

[40] C. Reul et al. “OCR4all - an Open-Source Tool Providing a (Semi-)Automatic OCR Workflow for Historical Printings”. In: *CoRR* abs/1909.04032 (2019). arXiv: 1909.04032 [cs].

[41] D. C. Schneider, M. Block, and R. Rojas. “Robust Document Warping with Interpolated Vector Fields”. In: *Proceedings of the 9th International Conference on Document Analysis and Recognition (ICDAR 2007)*. Curitiba, Paraná, Brazil: IEEE Computer Society, Sept. 2007, pp. 113–117.

[42] R. Sedgewick and K. Wayne. *Algorithms*. Fourth. Addison-Wesley Professional, 2011. ISBN: 0-321-57351-X.

[43] F. Shafait. “Geometric Layout Analysis of Scanned Documents”. Dissertation. Technical University of Kaiserslautern, Germany, Apr. 2008.

[44] D. Smith and R. Cordell. *A Research Agenda for Historical and Multilingual Optical Character Recognition*. Northeastern University, 2018.

[45] R. Smith. “An Overview of the Tesseract OCR Engine”. In: *Proceedings of the 9th International Conference on Document Analysis and Recognition (ICDAR 2007)*. Vol. 2. Curitiba, Paraná, Brazil, Sept. 2007, pp. 629–633.

[46] P. Ströbel and S. Clematide. *Improving OCR of Black Letter in Historical Newspapers: The Unreasonable Effectiveness of HTR Models on Low-Resolution Images*. Utrecht, The Netherlands, July 2019. DOI: 10.5167/uzh-177164. URL: https://dev.clariah.nl/files/dh2019/boa/0694.html.

[47] S. Suzuki and K. Abe. “Topological Structural Analysis of Digitized Binary Images by Border Following”. In: *Computer Vision, Graphics, and Image Processing* 30 (1985), pp. 32–46.

[48] S. Tanner, T. Muñoz, and P. H. Ros. “Measuring Mass Text Digitization Quality and Usefulness: Lessons Learned from Assessing the OCR Accuracy of the British Library’s 19th Century Online Newspaper Archive”. In: *D-Lib Magazine* 15.7/8 (July 2009). ISSN: 1082-9873.

[49] M. C. Traub, J. van Ossenbruggen, and L. Hardman. “Impact Analysis of OCR Quality on Research Tasks in Digital Archives”. In: *Research and Advanced Technology for Digital Libraries*. Ed. by S. Kapidakis, C. Mazurek, and M. Werla. Cham: Springer International Publishing, 2015, pp. 252–263. ISBN: 978-3-319-24592-8.

[50] D. van Strien et al. “Assessing the Impact of OCR Quality on Downstream NLP Tasks:” in: *Proceedings of the International Conference on Agents and Artificial Intelligence (ICAART 2020)*. Valletta, Malta: Science and Technology Publications, 2020, pp. 484–496. ISBN: 978-989-758-395-7. DOI: 10.5220/0009169004840496.

[51] H. Walravens, ed. *The Impact of Digital Technology on Contemporary and Historic Newspapers*. Berlin, Boston: De Gruyter Saur, 2008. ISBN: 978-3-598-44126-4. DOI: 10.1515/9783598441264.

[52] C. Wick, C. Reul, and F. Puppe. “Calamari - a High-Performance Tensorflow-Based Deep Learning Package for Optical Character Recognition”. In: *CoRR* abs/1807.02004 (2018). arXiv: 1807.02004 [cs].

[53] M. Wu et al. “A Model Based Book Dewarping Method to Handle 2D Images Captured by a Digital Camera”. In: *Proceedings of the 9th International Conference on Document Analysis and Recognition (ICDAR 2007)*. Vol. 1. Curitiba, Paraná, Brazil, Sept. 2007, pp. 158–162.

[54] C. E. Wulfman et al. *Complexities in the Use, Analysis, and Representation of Historical Digital Periodicals*. Utrecht, The Netherlands, July 2019. DOI: doi:10.34894/WTP281. URL: https://dataverse.nl/api/access/datafile/19099.

[55] T.-I. Yang, A. J. Torget, and R. Mihalcea. "Topic Modeling on Historical Newspapers". In: *Proceedings of the 5th ACL-HLT Workshop on Language Technology for Cultural Heritage, Social Sciences, and Humanities*. Portland, OR, USA: Association for Computational Linguistics, June 2011, pp. 96–104.

[56] X. Zhong, E. ShafieiBavani, and A. Jimeno-Yepes. "Image-Based Table Recognition: Data, Model, and Evaluation". In: *CoRR* abs/1911.10683 (2019). arXiv: 1911.10683 [cs].