From Text to Data
Digitization, Text Analysis and Corpus Linguistics

Patrick Jentsch, Stephan Porada

1. Introduction

Working with sources like books, protocols and other documents is the basis of most scientific work and research in the humanities. Unfortunately, most of these sources are only available on paper or come in other analog forms like parchments.

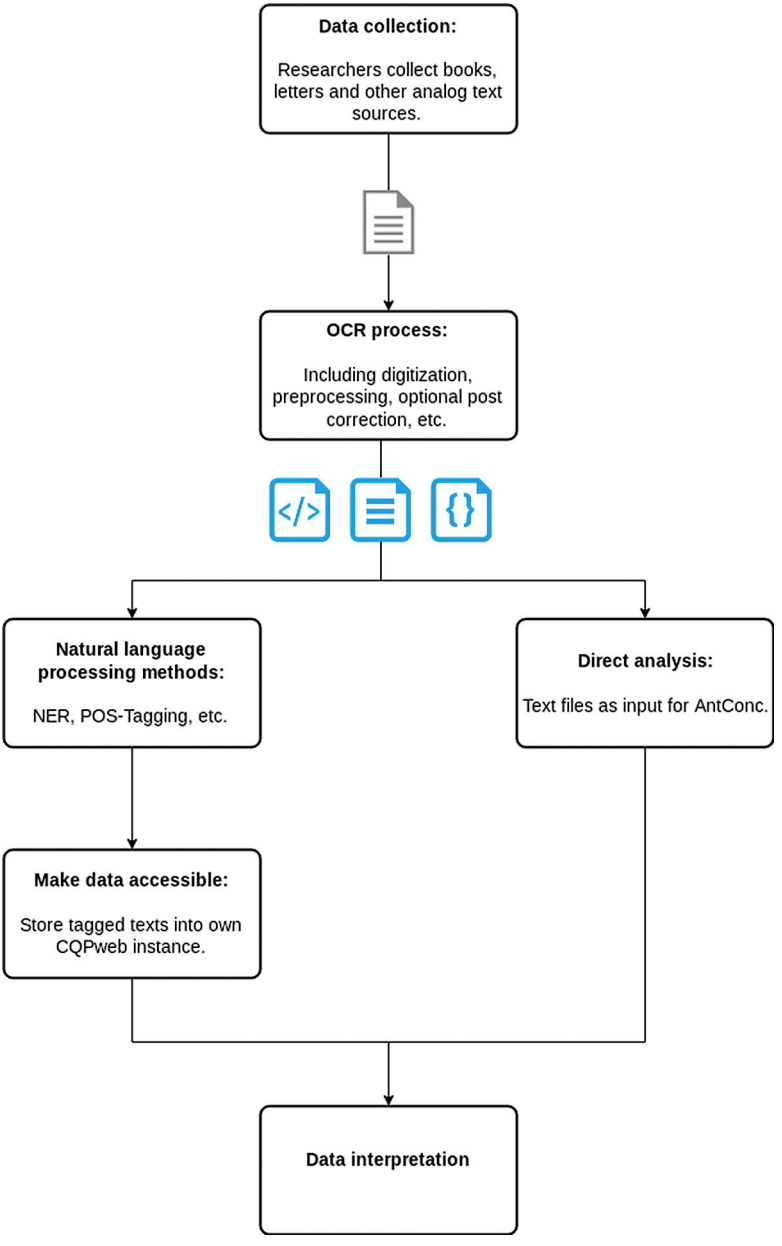
Within the field of Digital Humanities methods like data mining, text analysis and corpus linguistics are widely spread and used. To apply these methods to current historical research, historic text sources have to be digitized and turned into machine readable data by following the main steps outlined below.

At first, sources must be scanned to create digital representations of them. Following this, their images are used as input for our optical character recognition (OCR) pipeline, which produces plain text data. This text can then be further analyzed by means of the methods mentioned above. The analysis involves several natural language processing methods (NLP) which will also be discussed. Figure 1 shows the overall process from text to data, namely the main steps of data collection, OCR processing, data analysis and data interpretation.

The goal of this article is to explain the technologies and software used by the INF team (Data Infrastructure and Digital Humanities) of the Collaborative Research Center (SFB) 1288 “Practices of Comparing” to turn historical documents into digitized text and thus create the data basis for further research steps including text analysis and corpus linguistics.

In part two we present arguments advocating the use of free and open source software (FOSS). Part three is an overview of the basic software and

Fig. 1: Flow chart showing the entire process from text to data



technologies used to implement and to deploy our pipelines to production. The fourth part is a detailed description of the OCR pipeline, its software and internal processes. The last part discusses the different natural language processing (NLP) and computer linguistic methods which can be applied to the output texts of the OCR pipeline. Most of these methods are implemented by using *spaCy*.

The source code of the pipelines described in part four and five can be downloaded from the Bielefeld University's *GitLab* page.[1]

Both repositories include detailed instructions for the installation and usage of both pipelines.

2. Free and open-source software (FOSS)

One of the main goals besides turning text into data is to only use software that meets specific criteria in terms of sustainability, longevity and openness. These selection criteria are based on the "Software Evaluation Criteria-based Assessment"[2] guideline published by the Software Sustainability Institute.[3] The latter helped us decide on which software suited our needs best. The following paragraphs show and explain some of our main selection criteria.

---

1 The repository https://gitlab.ub.uni-bielefeld.de/sfb1288inf/ocr/tree/from_text_to_data contains the OCR pipeline. The repository https://gitlab.ub.uni-bielefeld.de/sfb1288inf/nlp/tree/from_text_to_data contains the NLP pipeline used for POS (part-of-speech) tagging, NER (named entity recognition) tagging, etc. [accessed: 31.08.2019].

2 *Jackson, Mike/Crouch, Steve/Baxter, Rob*, Software Evaluation: Criteria-Based Assessment (Software Sustainability Institute, November 2011), https://software.ac.uk/sites/default/files/SSI-SoftwareEvaluationCriteria.pdf [accessed: 31.08.2019].

3 This guideline again is based on the *ISO/IEC 9126-1* standard. The standard has been revised and was replaced in 2011 by the new *ISO/IEC 25010:2011* standard. *International Organization for Standardization*, ISO/Iec 9126-1:2001: Software Engineering – Product Quality – Part 1: Quality Model (International Organization for Standardization, June 2001), https://www.iso.org/standard/22749.html [accessed: 31.08.2019] and *International Organization for Standardization*, ISO/Iec 25010:2011: Systems and Software Engineering – Systems and Software Quality Requirements and Evaluation (SQuaRE) – System and Software Quality Models (International Organization for Standardization, March 2011), https://www.iso.org/standard/35733.html [accessed: 31.08.2019].

Sustainability, maintainability and usability ensure that every step in the process of turning text into data is documented and therefore reproducible. The main selection criteria consist of different subcriteria. To assess the sustainability and maintainability of software, for example, questions regarding copyright and licensing have to be answered.[4] Choosing free and open-source software (FOSS) ensures that the source code of those tools can always be traced. Processing steps conducted with FOSS are therefore always documented and reproducible. Another example of a subcriterion is interoperability.[5] The main aim of this criterion is to ensure that the software is easily interoperable with other software. In our case we mainly wanted to ensure that every software produces data output in open standards like XML or just plain text. This is crucial because output data created by one application or one software has to be easily usable with other software. In addition, open formats like XML are user-friendly and can easily be read for first evaluations of the data. Therefore, it is best practice to publish data in formats like XML because the scientific community can easily review, use and alter the data.

Conducting a criteria based software assessment using the main and subcriteria mentioned above naturally leads to only or mainly choosing and using FOSS.

In addition to the already mentioned advantages, this process ensures that only software is chosen that can be used over longer periods of time, even beyond the actual project phase. This is necessary because software and services which are based on a particular program still have to be usable after the end of the project phase.

3. Basic software

Our software implementations are based on some basic technologies. This part gives a brief introduction to this software.

4 *M. Jackson/S. Crouch/R. Baxter,* Software Evaluation, 7-8.

5 Ibid., 13.

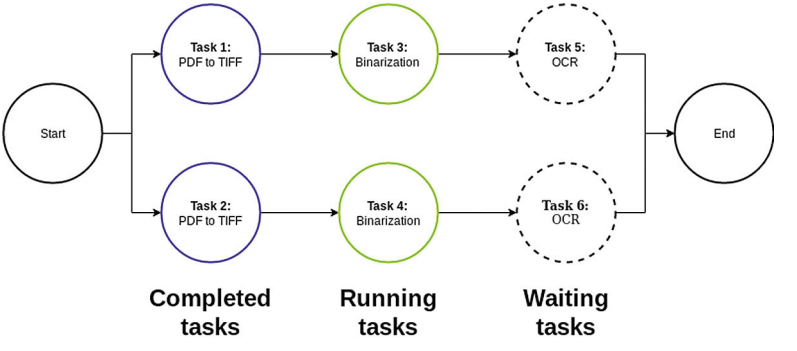
pyFlow

The OCR and NLP modules we developed were implemented by using the python module *pyFlow*.[6] It is currently only available for Python 2.[7]

This module is used to manage multiple tasks in the context of a task dependency graph. In general this means that pyFlow creates several tasks with hierarchical dependencies (upstream tasks have to be completed before the dependent task will be executed). Tasks with the same satisfied dependency (or tasks with no first dependencies) can be completed concurrently. Other tasks depending on those to be finished first will only be executed if the necessary dependencies are satisfied. Figure 2 shows an example of a simple OCR process.

At first, the OCR pipeline has to convert input files from PDF to TIFF because the binarization, which constitutes the following preprocessing step, can only be done with TIFF files. The conversion of each PDF file to TIFF is a single task. Because *pyFlow* is designed with parallelization in mind, some tasks can be executed concurrently. In this case, the conversion from PDF to TIFF can be done for multiple files simultaneously depending on the available RAM and CPU cores.[8] As soon as all PDFs are converted, the binarization of the files can start.

Fig. 2: Dependency graph example for a simple OCR process



6 *Saunders, Chris*, PyFlow: A Lightweight Parallel Task Engine, version 1.1.20, 2018, https://github.com/Illumina/pyflow/releases/tag/v1.1.20 [accessed: 31.08.2019].

7 Ibid.

8 Ibid.

The entire task management is automatically done by *pyFlow*. We just have to define the workflows in Python code and the engine handles all task dependencies and parallelization. Using *pyFlow* helps us to handle huge data inputs of several input files. The entire workload is automatically distributed among the maximum number of available CPU cores, which accelerates the entire process.

Container virtualization with Docker

For our development and production environment we decided to deploy our Optical Character Recognition and natural language processing software in containers. For that purpose, we use a container virtualization software called *Docker*.[9] With Docker it is possible to easily create and use Linux containers.

> "A Linux® container is a set of one or more processes that are isolated from the rest of the system. All the files necessary to run them are provided from a distinct image, meaning that Linux containers are portable and consistent as they move from development, to testing, and finally to production. This makes them much quicker than development pipelines that rely on replicating traditional testing environments. Because of their popularity and ease of use containers are also an important part of IT security."[10]

In order to get a container up and running it is necessary to build a so called container image. An image is used to load a container in a predefined state at startup. Thus the image represents the initial state of a container including the chosen operating system base, software installations and configurations. It freezes a software deployment in its creation state and because of its portability it can then be shared easily.[11] That is why it is also suitable for publishing a software deployment in the context of a publication like the one at hand.

---

9 *Docker*, version 18.09.1 (Docker, 2013), https://www.docker.com/ [accessed: 31.08.2019].

10 *Red Hat Inc.*, What's a Linux Container?, https://www.redhat.com/en/topics/containers/whats-a-linux-container [accessed: 20.05.2019].

11 *Boettiger, Carl*, An Introduction to Docker for Reproducible Research, in: ACM SIGOPS Operating Systems Review 49 (2015), 71–79, https://doi.org/10.1145/2723872.2723882.

To create a container image, Docker provides a build system which is based on so called *Dockerfiles*. These files act as a blueprint for the image the user wants to create. In order to create and maintain it, the developer only needs knowledge about operating a terminal and the concepts of the operating system used within the image.

Our images are based on the free and open source Linux distribution *Debian 9*.[12] We ensured that all our software that is installed on top of this basis is also free and open source software.

4. A practical approach to optical character recognition of historical texts

As mentioned in the introduction, the first step of creating data is turning historical texts and sources into machine readable formats. The OCR process creates text files, namely XML, hOCR and PDF (with text layer). This process will be described in this part in detail. The corresponding source code and documentation can be downloaded from the GitLab page.[13] We mainly use the text files for further natural language processing steps which are described later in the text.

The entire process of turning books as well as other sources into data can be divided into a few manual pre- and postprocessing steps. The actual OCR is done automatically by our OCR pipeline.

Before the actual steps are described, we will briefly outline the goals of our pipeline and the software used. We also provide a short summary of the history of the OCR engine Tesseract.

Goals of our OCR pipeline: handling middle to large scale text input

This article proposes a practical way to do mid to large scale OCR for historic documents with a self developed Tesseract based pipeline. The pipeline is a tool to easily create mid to large sized text corpora for further research.

12 *Debian*, version 9 (The Debian Project, 2017), https://www.debian.org/ [accessed: 31.08.2019].

13 The code for the OCR pipline especially the pyFlow part is based on the original work of Madis Rumming, a former member of the INF team.

For now, the pipeline is a command line-based application which can be used to subject input documents to optical text recognition with a few simple commands. As of yet it is only used by the INF team of the SFB. Researchers have to request the OCR process for their sources and documents via our internal ticket system.

In the future, researchers will be able to upload any digitized TIFF or PDF document to the pipeline. It can handle multiple input documents, for example books, letters or protocols, at the same time and will automatically start the OCR process. Those documents will then be turned into text data. Researchers can choose between different languages per pipeline instance but not per document input.

To achieve this goal the INF team will build a virtual research environment (VRE).[14] The VRE will be implemented as a web application which can be easily used by every researcher of the SFB1288. Besides starting OCR processes researchers will also be able to start the tagging processes of text files. Different tagging sets will be available like the ones discussed in this article. Finally, researches will also be able to import tagged texts into an information retrieval system. We plan to either implement CQPweb[15] or use some of the provided application programming interfaces (APIs) to build our own front end.[16]

By providing this VRE the INF team will provide the researchers of the SFB1288 with many different tools which will aid them during different research steps.

Implementation

All software dependencies needed to run our pipeline are documented in our source code repository hosted by the GitLab system of the Bielefeld University Library.[17]

14 The development of this VRE is in an early stage.

15 CQPweb is a web-based graphical user interface (GUI) for some elements of the IMS Open Corpus Workbench (CWB). The CWB uses the efficient query processor CQP, *Hardie, Andrew/Evert, Stefan*, IMS Open Corpus Workbench, http://cwb.sourceforge.net/ [accessed: 31.08.2019].

16 *Sourceforge*, CWB/Perl & Other APIs, http://cwb.sourceforge.net/doc_perl.php [accessed: 13.05.2019].

17 The source code, documentation and pre-built images can be accessed here: https://gitlab.ub.uni-bielefeld.de/sfb1288inf/ocr [accessed: 31.08.2019].

The pipeline consists of three files. The main file is *ocr* which implements the actual OCR pipeline. The file *prase_hocr* is used to create a valid *DTA-Basisformat*[18] XML from the hOCR output files which follows the P5 guidelines of the Text Encoding Initiative (TEI).[19] The script *parse_hocr* is called by the main file *ocr*. The last and third file is the Dockerfile used to automatically create an image. The image can then be used to start multiple containers to run multiple OCR processes. Detailed installation instructions can be found in the documentation.

A short history of Tesseract

We use *Tesseract* for the actual OCR process. Tesseract is an open source OCR engine and a command line program. It was originally developed as a PhD research project at Hewlett-Packard (HP) Laboratories Bristol and at Hewlett-Packard Co, Greeley Colorado between 1985 and 1994.[20] In 2005 HP released Tesseract under an open source license. Since 2006 it has been developed by Google.[21]

The latest stable version is 4.0.0, which was released on October 29, 2018. This version features a new long short-term memory (LSTM)[22] network based

18 *Berlin-Brandenburgische Akademie der Wissenschaften*, Ziel und Fokus des DTA-Basisformats (Deutsches Textarchiv, Zentrum Sprache der Berlin-Brandenburgischen Akademie der Wissenschaften), http://www.deutschestextarchiv.de/doku/basisformat/ziel.html#topic_ntb_5sd_qs__rec [accessed: 12.03.2019].

19 *Text Encoding Initiative*, TEI P5: Guidelines for Electronic Text Encoding and Interchange (TEI Consortium) https://tei-c.org/release/doc/tei-p5-doc/en/Guidelines.pdf [11.06.2019].

20 *Smith, Ray*, An Overview of the Tesseract OCR Engine, in: Ninth International Conference on Document Analysis and Recognition (ICDAR 2007) Vol 2 (2007), https://doi.org/10.1109/icdar.2007.4376991 [accessed: 31.08.2019].

21 *Google Inc.*, Tesseract OCR (2019), https://github.com/tesseract-ocr/tesseract/ [accessed: 31.08.2019].

22 A special kind of recurrent networks, capable of using context sensitive information which is not near to the data which is processed (long-term dependencies). A LSTM network can be used to predict words in a text with respect of information which is further away. For example in a text it says that someone is from France and way later it says that this person speaks fluently x. Where x (= french) is the word to be guessed with the LSTM network by using the first information. *Olah, Christopher*, "Understanding LSTM Networks, https://colah.github.io/posts/2015-08-Understanding-LSTMs/ [accessed: 26.03.2019].

OCR engine which is focused on line recognition.[23] Our own developed pipeline is based on the upcoming minor release, specifically on version 4.1.0-rc1 to benefit from the new neural net technology. During the development we also used version 3.05.01.

Choosing data files for Tesseract

Tesseract needs models or so called data files per language for the OCR process. Models are manually trained. There are three main sets of trained data/data files for various languages available.

1. tessdata (Legacy models for Version 3)
2. tessdata_fast (Fast standard models)
3. tessdata_best (Slower for slightly better accuracy)

For now we are using the following models for the corresponding languages:

- German model from tessdata_best[24]
- German Fraktur not available as tessdata_best[25]
- English model from tessdata_best[26]
- English middle from tessdata_best[27]
- French from tessdata_best[28]
- French middle from tessdata_best[29]
- Portuguese from tessdata_best[30]
- Spanish from tessdata_best[31]

23 *Google Inc.*, Tesseract OCR.

24 https://github.com/tesseract-ocr/tessdata_best/raw/master/deu.traineddata

25 https://github.com/tesseract-ocr/tessdata/raw/master/deu_frak.traineddata

26 https://github.com/tesseract-ocr/tessdata_best/raw/master/eng.traineddata

27 https://github.com/tesseract-ocr/tessdata_best/raw/master/enm.traineddata

28 https://github.com/tesseract-ocr/tessdata_best/raw/master/fra.traineddata

29 https://github.com/tesseract-ocr/tessdata_best/raw/master/frm.traineddata

30 https://github.com/tesseract-ocr/tessdata_best/raw/master/por.traineddata

31 https://github.com/tesseract-ocr/tessdata_best/raw/master/spa.traineddata

We aim to only use trained data from tessdata_best to achieve high quality OCR results. Only German Fraktur is not available as a tessdata_best model.

Overview of the entire OCR process and the pipeline

This part describes the function of the developed pipeline in detail, beginning with the digitization and preprocessing of input documents. Following these steps, the actual process of OCR is described in general to provide an overview of the underlying principles and technologies used. Lastly, the output files of the pipeline are described, and we explain why those files are generated and what they are used for.

In addition to this we also discuss the accuracy of the OCR and how it can affect the text data output as well as further research using the data.

Figure 3 shows the entire OCR process including manual and automatic steps. Every step is discussed in the following parts.

Input for the pipeline: digitization and collection of historic documents

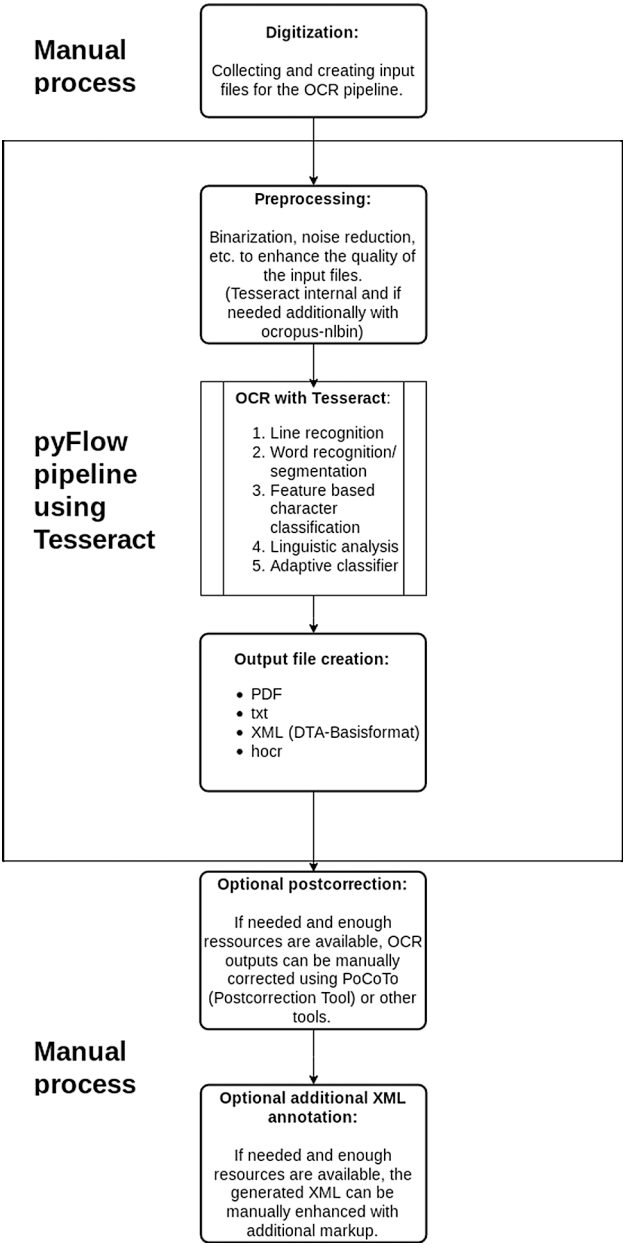
As mentioned above the pipeline accepts TIFF (only multi-page TIFFs per document) and PDF files as input. These input files have to be obtained or created first. The process of creating input files as discrete sets of pixels from physical media like paper based books, etc. is called digitization.[32] Scanning a book and creating a PDF file from it is only one example of digitization, however. Another example is taking a picture of a document with the camera of a mobile phone, which creates a JPEG file. Both examples obviously result in digital representations of different quality.

Digitization is the first step of the full OCR process as depicted in figure 3.

Below we describe the most common ways how researchers of the SFB are obtaining or creating input files for the pipeline. The different ways result in different qualities of input files.

32 *Ye, Peng/Doermann, David*, Document Image Quality Assessment: A Brief Survey, in: 2013 12th International Conference on Document Analysis and Recognition (2013), 723, https://doi.org/10.1109/icdar.2013.148.

Fig. 3: The entire OCR process



The first and most common way to obtain input files for the pipeline is to use digitized documents provided by libraries or other institutions. These files vary in quality depending on how they were created.

It is also common to obtain image files from libraries which were created from microfilms. This way is similar to obtaining scans of books like mentioned above. Unfortunately, sometimes those images are of poor quality.

Our experience has shown that images obtained from libraries do not meet our quality demands sometimes. Whenever this is the case and a physical copy of said document is at hand, we repeat the digitization within our own quality parameters.

We advise to always assess the quality of input images based on the criteria listed below. Especially background noise and geometric deformation decisively decrease the quality of the OCR process.

It may sound as if libraries in general do not do a very good job of creating high quality digital representations of books and other documents. This is not the case because we have to keep in mind that the digitization is mainly done with human readers in mind. Humans are far better at reading documents of relatively low quality than computers. The demand for high quality images used for OCR processes and thus for corpus linguistic projects has risen over the years. Most of the libraries are adapting to this new trend and are providing the necessary images.

The third way to obtain input files is to perform our own scans of books and microfilms. This should always be the method of choice if already obtained digitized input files are of low quality. In general, it is better to always perform own scans with predefined parameters to ensure the best possible quality.

During the stage of obtaining and creating input files we can already enhance the accuracy of the subsequent OCR process significantly. It should always be the goal to obtain or create image files of the highest quality. The higher the quality of the scans the better are the end results of the OCR, which ultimately leads to high quality text data corpora.

There are a few criteria on how to determine if a digital representation is of good quality:[33]

1) Stroke level
   a) Touching characters
   b) Broken characters
   c) Additive noise:
      i) Small speckle close to text (For example dirt.)
      ii) Irregular binarization patterns
2) Line level
   a) Touching lines
   b) Skewed or curved lines
   c) line inconsistency
3) Page level
   a) background noise:
      i) Margin noise
      ii) Salt-and-pepper
      iii) Ruled line
      iv) Clutter
      v) Show through & bleed through
      vi) Complex background binarization patterns
   b) Geometric deformation:
      i) Warping
      ii) Curling
      iii) Skew
      iv) Translation
4) Compression methods
   a) Lossless compression methods are preferred

When creating our own scans, we can follow some best practices outlined below to avoid some of the above mentioned problems with the digital representation of documents:

33 *P. Ye/D. Doermann*, Document Image Quality Assessment: A Brief Survey, 724.

General best practices are mainly based on the official Tesseract wiki entry:[34]

- Create scans only with image scanners. Other digitization measures like using mobile phone cameras etc. are not advised. These will easily introduce geometric deformation, compression artifacts and other unwanted problems all mentioned in this list and the criteria list above. We mention this especially because we often had to handle images of books that have been taken with mobile phone cameras or other cameras.
- Avoid creating geometric deformation like skewing and rotation of the page. (This can be hard with thick books because the book fold will always introduce some warping.)
- Avoid dark borders around the actual page. (These will be falsely interpreted as characters by Tesseract.)
- Nonetheless also avoid not having any border or margin around the text.
- Avoid noise like show through, bleed through etc.
- One page per image recommended. Do not use double-sided pages.

Technical specifications also mainly based on the official Tesseract wiki entry:[35]

1. Scan with 300 dots per inch (DPI)
2. Use lossless compression
   a) Avoid JPEG compression methods. This method introduces compression artifacts and compression noise around the characters.[36] This is due to the discrete cosine transform (DCT) which is part of the JPEG compression method.[37] (Figure 4 shows an example of the differences between a lossless compressed TIFF and a lossy compressed jpeg.)
   b) Avoid lossy compression methods in general.

34 *Google Inc.*, ImproveQuality: Improving the Quality of the Output (2019), https://github.com/tesseract-ocr/tesseract/wiki/ImproveQuality [accessed: 31.08.2019].

35 Ibid.

36 *P. Ye/D. Doermann*, Document Image Quality Assessment, 723.

37 *Oztan, Bazak, et al.*, Removal of Artifacts from JPEG Compressed Document Images, in: Reiner Eschbach/Gabriel G. Marcu (eds.), Color Imaging XII: Processing, Hardcopy, and Applications, (SPIE) 2007, 1-3, https://doi.org/10.1117/12.705414.

c) This is why we use and recommend TIFF files with lossless compression using the Lempel-Ziv-Welch-Algorithm (LZW-Algorithm or LZW)[38]
d) We also accept PDFs. (We have to admit that this is a trade off for convenience. PDFs can be of the same quality as TIFF files if they are created from images using the FLATE/LZW compression. The default though is lossy JPEG compression.)[39]

Fig. 4: Lossless and lossy image compression

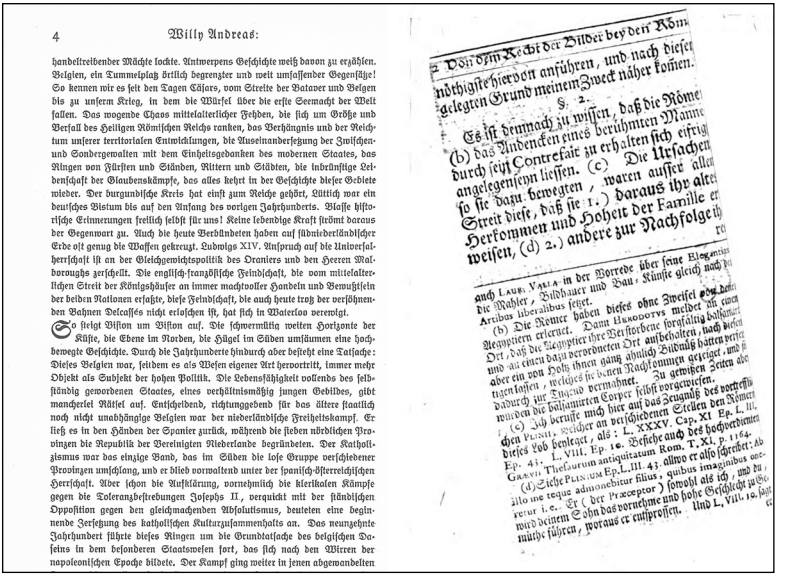
The top string shows a lossless compressed TIFF file. The lower string shows a lossy compressed JPEG file. Both files are binarized. The JPEG compression rate is 70 to give a better visual example. (self-created)

To sum up, digitization of historical documents is already an important step significantly influencing the accuracy of the OCR process. In this part we have outlined how we obtain and create input files for our OCR pipeline. The main goal is to always obtain or create input files of the highest quality as possible. To achieve this, we described criteria to determine the quality of given input files. Additionally, we described some rules on how to create high quality scans of historic documents.

38 *Adobe Systems Incorporated*, TIFF: Revision 6.0, version 6.0 (1992), 57–58, https://www.adobe.io/content/dam/udp/en/open/standards/tiff/TIFF6.pdf [accessed: 31.08.2019].

39 *Adobe Systems Incorporated*, Document Management – Portable Document Format – Part 1: PDF 1.7 (2008), 25 ff., https://www.adobe.com/content/dam/acom/en/devnet/pdf/PDF32000_2008.pdf [accessed: 26.03.2019].

Fig. 5: Comparison between high and low quality



The left part of the figure shows a high quality scan. The right part shows a low quality scan with several problems like background noise and geometric deformation. (self-created)

Starting the pipeline with input files and user set parameters

In the next step the collected and created files are passed into the actual OCR pipeline. The OCR pipeline is written in python using the pyFlow package. It is deployed by using Docker. A description of Docker and pyFlow can be found in part 3.

As soon as this is done, we can place our input files in the folder "files_for_ocr", which was created during the setup of the pipeline. Files must be sorted into folders beforehand on a per document basis. For example, a scanned book results in one multi-page TIFF file. This file has to be placed into its own corresponding folder inside the folder *files_for_ocr*. PDFs should be handled the same way. We can put as many folders into the input folder of the pipeline as we want. The only constraint here is that every input document should be of the same language because we tell Tesseract to use a specific language model for the OCR process. Because the pipeline is written using pyFlow it creates different processes for each document in the input folder and works

through those in an efficient manner (see also part 3.). To handle multiple documents of different languages at the same time we recommend starting a new Docker instance per language and place the documents per language in the corresponding input folder of the Docker instance. If a document consists of multiple languages it is possible to tell Tesseract which language models it should use at the same time.

Once every input file has been put into the folder *files_for_ocr* we can start the pipeline with a simple command. The command and further examples can be found in our corresponding GitLab repository documentation.[40]

The following parameters can be set by the user:

Language
This parameter tells Tesseract which model it should use for the OCR process. Language should be set to the corresponding language of the input files.

Binarization
The user can decide to binarize the input images with ocropus-nlbin in an additional upstream preprocessing step (see below for more information on binarization).

Pipeline processing step 1: unify input files

Before the preprocessing of the input files starts, the pipeline converts PDF files into TIFF files using the package *pdftoppm*. Every page of the PDF is converted into one TIFF file with 300 DPI using the lossless LZW compression method. Multi-page TIFF files are split per page into individual files. Now that all input files are of the same type every following preprocessing step (e. g., binarization) can be applied to those uniformly file by file.

Pipeline processing step 2: preprocessing of the input files

The first internal step of the pipeline is preprocessing of the input images. Some major steps in enhancing the image quality are described below.

40 *Jentsch, Patrick/Porada, Stephan*, Docker Image: Optical Character Recognition, version 1.0, Bielefeld 2019, https://gitlab.ub.uni-bielefeld.de/sfb1288inf/ocr/container_registry [accessed: 31.08.2019].

Binarization and noise removal

One important first step in preprocessing is binarization. The goal of binarization is to reduce the amount of noise and useless information in the input image.[41]

In general binarization is the step of converting a color image into a black and white image. The idea is to only extract the pixels which actually belong to the characters and discard any other pixel information which, for example, is part of the background. To achieve this the technique of thresholding is used. Basically, the method of thresholding analyses each pixel of a given picture and compares its grey level or another feature to a reference value. If the pixels value is below the threshold it will be marked as a black pixel and thus as belonging to a character. If the value of the pixel is above the threshold it will be labeled as white pixel und thus be handled as the background. Binarization techniques using thresholding can be divided into two classes: global and local thresholding. Both methods differ in what reference value for the pixel comparison is being used. Global thresholding calculates one reference value per pixel in one picture while local thresholding calculates the reference value for each pixel based on the neighboring pixels.[42] Figure 6 shows the successfully applied binarization step done with ocropus-nlbin.

The pipeline will always use the built in Tesseract binarization. Tesseract's built in binarization uses the Otsu algorithm.[43] There is also the option to binarize the pictures before passing them to Tesseract, if the built in binarization is not sufficient. For this additional upstream binarization process our pipeline uses the ocropus-nlbin library. This step can easily be invoked by using the corresponding parameter (see our documentation).

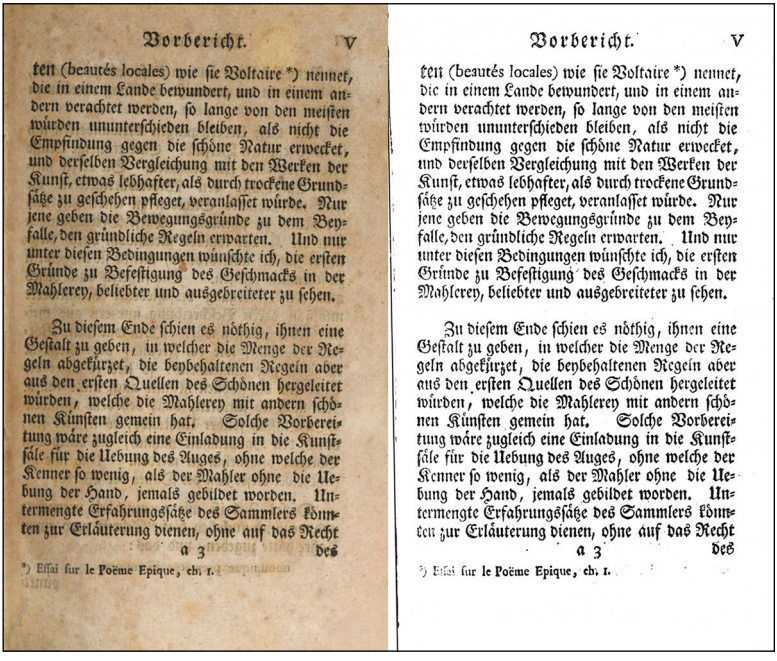
Omitting or explicitly using this additional upstream step can in both cases either result in better or worse accuracy. Which option is chosen is

41 *Chaudhuri, Arindam, et al.*, Optical Character Recognition Systems for Different Languages with Soft Computing, Springer International Publishing: 2017, 90–92, 17–22, https://doi.org/10.1007/978-3-319-50252-6.

42 *Cheriet, Mohamed*, et al., Character Recognition Systems, John Wiley & Sons, Inc.: 2007, 8–15, https://doi.org/10.1002/9780470176535.

43 *Google Inc.*, ImproveQuality: Improving the Quality of the Output.

Fig. 6: Binarization process



Vorbericht. V

ten (beautés locales) wie sie Voltaire *) nennet, die in einem Lande bewundert, und in einem andern verachtet werden, so lange von den meisten würden ununterschieden bleiben, als nicht die Empfindung gegen die schöne Natur erwecket, und derselben Vergleichung mit den Werken der Kunst, etwas lebhafter, als durch trockene Grundsätze zu geschehen pfleget, veranlasset würde. Nur jene geben die Bewegungsgründe zu dem Beyfalle, den gründliche Regeln erwarten. Und nur unter diesen Bedingungen wünschte ich, die ersten Gründe zu Befestigung des Geschmacks in der Mahlerey, beliebter und ausgebreiteter zu sehen.

Zu diesem Ende schien es nöthig, ihnen eine Gestalt zu geben, in welcher die Menge der Regeln abgekürzet, die beybehaltenen Regeln aber aus den ersten Quellen des Schönen hergeleitet würden, welche die Mahlerey mit andern schönen Künsten gemein hat. Solche Vorbereitung wäre zugleich eine Einladung in die Kunstsäle für die Uebung des Auges, ohne welche der Kenner so wenig, als der Mahler ohne die Uebung der Hand, jemals gebildet worden. Untermengte Erfahrungssätze des Sammlers könten zur Erläuterung dienen, ohne auf das Recht

a 3 des

*) Essai sur le Poëme Epique, ch. 1.

Vorbericht. V

ten (beautés locales) wie sie Voltaire *) nennet, die in einem Lande bewundert, und in einem andern verachtet werden, so lange von den meisten würden ununterschieden bleiben, als nicht die Empfindung gegen die schöne Natur erwecket, und derselben Vergleichung mit den Werken der Kunst, etwas lebhafter, als durch trockene Grundsätze zu geschehen pfleget, veranlasset würde. Nur jene geben die Bewegungsgründe zu dem Beyfalle, den gründliche Regeln erwarten. Und nur unter diesen Bedingungen wünschte ich, die ersten Gründe zu Befestigung des Geschmacks in der Mahlerey, beliebter und ausgebreiteter zu sehen.

Zu diesem Ende schien es nöthig, ihnen eine Gestalt zu geben, in welcher die Menge der Regeln abgekürzet, die beybehaltenen Regeln aber aus den ersten Quellen des Schönen hergeleitet würden, welche die Mahlerey mit andern schönen Künsten gemein hat. Solche Vorbereitung wäre zugleich eine Einladung in die Kunstsäle für die Uebung des Auges, ohne welche der Kenner so wenig, als der Mahler ohne die Uebung der Hand, jemals gebildet worden. Untermengte Erfahrungssätze des Sammlers könten zur Erläuterung dienen, ohne auf das Recht

a 3 des

*) Essai sur le Poëme Epique, ch. 1.

The left part shows an input image before the binarization step has been applied. The right part shows the same image after the binarization step. Note that during the binarization minimal skew is also automatically removed. (self-created)

decided on a per input file basis taking the input files quality into account. Sometimes we also evaluate the output accuracy of one step and rerun the OCR process to achieve a better accuracy by using or not using ocropus-nlbin. Subchapter Accuracy in Part 5 discusses the different accuracy values for the use and non-use of the additional upstream binarization step for some example files.

Skew detection and correction

Even when using scanners for the digitization process, there will always be a few degrees of tilt or skew of the text due to human influence (for example how the book was placed inside the scanner etc.).

Tesseract's line finding algorithm was designed to work without having to deskew a page to successfully identify text lines. This design choice avoids the loss of image quality.[44,45] Manual deskewing is only needed if the skew is too severe, as mentioned in the official wiki.[46] According to the original paper from 1995 describing the algorithm[47] the line finding algorithm produces robust results for angles under 15 degrees. Because of this we have not implemented automatic deskewing of input files.

If the skew of the input files is too severe, they have to be deskewed manually before passing them into the pipeline. Manual deskewing is also advised because the skew can vary severely from page to page. Automatic deskewing could therefore result into the loss of text parts, depending on the discrepancies in skew between pages.

After the preprocessing steps, the actual OCR process starts. The process is described in the following part.

Pipeline processing step 3: OCR process

This part gives an overview of the internal steps of the actual OCR process. We describe which steps are performed by Tesseract internally to perform the OCR process. Some of the steps are done by various other OCR engines in general, some of the steps are specific to Tesseract. Those differences are highlighted and explained.

One of the first steps performed by Tesseract is line finding.[48] This step is specific to Tesseract because the algorithm was explicitly designed for it.[49] In general this step detects lines of text in already provided and identified text regions. One advantage of the algorithm is its achievement of robust results for line recognition on pages with a skew of up to 15 degrees.

---

44 *R. Smith*, An Overview of the Tesseract OCR Engine.

45 *Smith, Ray*, A Simple and Efficient Skew Detection Algorithm via Text Row Accumulation, in: Proceedings of 3rd International Conference on Document Analysis and Recognition (IEEE Comput. Soc. Press, 1995), https://doi.org/10.1109/icdar.1995.602124.

46 *Google Inc.*, ImproveQuality: Improving the Quality of the Output.

47 *R. Smith*, A Simple and Efficient Skew Detection Algorithm via Text Row Accumulation.

48 *R. Smith*, An Overview of the Tesseract OCR Engine.

49 *R. Smith*, A Simple and Efficient Skew Detection Algorithm via Text Row Accumulation.

Once the lines have been identified, additional steps are being executed to fit the baseline of every line more precisely. Tesseract handles curved lines especially well.[50] This is another advantage of the algorithm as curved lines are a common artifact in scanned documents.

The next major step executed by Tesseract is word recognition. This step is also performed by various other available OCR engines. The goal of word recognition is to identify how a word should be segmented into characters.[51,52] For this step characters have to be recognized and then chopped or segmented.

Another part of the Tesseract OCR process is the so called *Static Character Classifier* or, in more general terms, the character classification. This step is feature based which is a common approach in various available OCR engines.[53,54] The goal of feature extraction is to identify essential characteristics of characters.[55] Based on the extracted features the classification process will be executed. Each character will be identified based on its features and will be assigned to its corresponding character class.[56,57]

One of the last steps Tesseract executes is a linguistic analysis. It only uses a minimal amount auf linguistic analysis.[58] Tesseract, for example, compares every word segmentation with the corresponding top dictionary word. This process is a statistical approach where the segmentation will be matched with the most likely corresponding word. This process is done for other categories besides the top dictionary word.[59]

These are the main steps done by Tesseract to turn images into machine readable text.

---

50 *R. Smith*, An Overview of the Tesseract OCR Engine.

51 *M. Cheriet et al.*, Character Recognition Systems, 204–206.

52 *R. Smith*, An Overview of the Tesseract OCR Engine.

53 Ibid.

54 *A. Chaudhuri et al.*, Optical Character Recognition Systems for Different Languages with Soft Computing, 28.

55 Ibid.

56 *A. Chaudhuri et al.*, Optical Character Recognition Systems for Different Languages with Soft Computing, 28.

57 *R. Smith*, An Overview of the Tesseract OCR Engine.

58 Ibid.

59 Ibid.

Pipeline processing step 4: output file creation

Tesseract can automatically create several output files after the OCR process is finished. Output files will be created per TIFF file. The file formats are:

hOCR
Standard output file. HTML based representation of the recognized text. Includes position of lines and characters matching the corresponding input image. Can for example be used to create PDFs with an image layer using the original input image. Also needed for post correction with PoCoTo. (See Pipeline processing step 5 in part 4.)

PDF
Tesseract automatically creates one PDF file per page consisting of an image layer and an invisible text layer. The image layer shows the input TIFF file. The text layer is placed in such a way that the recognized strings mach the actual visible text in the image.

Besides those two outputs the pipeline automatically creates the following files per input document:

- Combined PDF (combines the single PDF pages into one file per document)
- Combined text file (created from the combined PDF file)
- DTA-Basisformat XML (created from the hOCR files per document)

The combined PDF file is created from the single page PDFs containing the image and text layer. The combined PDF files are mainly created for humans because they are easily readable on any device.

The text file per document is created from the text output files of Tesseract. For that purpose we use a simple bash command which utilizes `cat`. We aim to export paragraphs and other formatting structures. The text files are mainly used as input for further computer linguistic methods. These methods are described in part 5.

Lastly, the pipeline automatically creates valid DTA-Basisformat XML files per input document. The XML files are created from the hOCR output files. The DTA-Basisformat XML structure follows the P5 guidelines of the

Text Encoding Initiative (TEI).[60] The DTA-Basisformat is developed by the Deutsches Textarchiv and recommended for digitizing and archiving historical texts.[61] One goal of the XML markup is to preserve the logical structure of the digitized texts. For example, headings, paragraphs and line breaks are being annotated with corresponding tags. Also, procedural markup of text color or italic written text is being annotated. The DTA-Basisformat syntax can also be used to annotate more uncommon text parts like poems, recipes, marginal notes or footnotes.

Creating the output files is the last automatic step done by the pipeline. Files can now be enhanced and corrected during postprocessing steps or be passed to the next process (POS tagging, NER etc.).

Pipeline processing step 5: optional manual postprocessing steps

If needed, manual postcorrection of the output texts can be done. For this, the post correction tool PoCoTo can be used. With this it is possible to use the hOCR files in conjunction with the corresponding TIFF files. PoCoTo gives the user a side by side view where one can compare the recognized text with the actual image representation. If the text does not match the image, the user can correct it. It is also possible to correct common repeated errors automatically and thus save time.

Alternatively, every common text editor can be used to correct the text in the hOCR files directly.

After the post correction, new PDFs, XML and text files have to be created manually.

Besides a simple postcorrection of the text, another manual and optional step would be the enhancement of the XML markup. Tesseract and our pipeline recognize paragraphs and line breaks, which are automatically written into the XML file. More sophisticated elements must be annotated by hand. The annotation can be done with any common text editor. Common elements that have to be annotated manually are marginal notes or footnotes.

60 *Text Encoding Initiative*, TEI P5: Guidelines for Electronic Text Encoding and Interchange.

61 *Berlin-Brandenburgische Akademie der Wissenschaften*, Ziel und Fokus des DTA-Basisformats.

Evaluation accuracy of Tesseract

In this part we talk about the accuracy and error rates of Tesseract and our pipeline. First, we briefly show some error rates and accuracy values for the Tesseract OCR engine published by Ray Smith working for Google.

Besides the official numbers we also show some results from our own accuracy tests performed with collected and self created test data as an input for our pipeline. The test data consists of digital input images and the corresponding manually created and corrected accurate textual content of those. In the context of OCR this transcription is called ground truth data.[62]

For these accuracy tests we will input the test data images into the pipeline and compare the output text with the ground truth text. From the discrepancies between the output and the ground truth data we can calculate two different error rates.

In the context of OCR evaluation two metrics are used to describe the error rate at two different levels: Character error rate (CER) and Word error rate (WER). Both metrics are calculated independently in regard to the length of the output text data. In order to achieve this, the number of mistakes is divided by the text length resulting in an error rate either for words or characters. This has to be done for both metrics.[63]

For the concrete calculation of those error rates we use the *OcrevalUAtion* tool.[64,65] This tool compares the ground truth text with the actual output text from the OCR pipeline and calculates the error rates accordingly.

The goal of our own accuracy tests is to see if our digitization, preprocessing and binarization steps are either beneficial or disadvantageous for the accuracy of the OCR process. We also want to measure the quality difference between files that have been binarized with ocropus-nlbin and those

62 *Carrasco, Rafael C.*, Text Digitisation, https://sites.google.com/site/textdigitisation/ [accessed: 11.06.2019], ch. 2.1.

63 Ibid.

64 *University of Alicante*, ocrevalUAtion, https://github.com/impactcentre/ocrevalUAtion [accessed: 05.04.2019].

65 *University of Alicante*, OcrevalUAtion, version 1.3.4 (2018), https://bintray.com/impactocr/maven/ocrevalUAtion [accessed: 31.08.2019].

that have not. In addition to that we can also compare our values to the ones published by Google.

We focus on the values for modern English and Fraktur.

Official numbers

The published accuracy results for Tesseract show quite low error rates for Latin languages like English and French. Tesseract version 4.0+ using the LSTM model and an associated dictionary check has an CER of 1.76 for English text. The WER is 5.77. The CER for French is 2.98 and the WER is 10.47. In general, the error rates for Latin languages are in a similar range.[66] Note that these low error rates are probably the result of high quality image inputs. We can deduce this from our own calculated error rates shown in the following part.

Own tests with ground truth data

Table 1 shows our own calculated error rates for different input data. We tested our pipeline with input images of two different quality levels. High quality images are TIFFs we created with our own scanners. These images were created with a minimum of 300 DPI and full color range. We also made sure that we introduced as little skew and rotation as possible during the scanning process. Input images of medium quality were created from PDF files with lower DPI and possible JPEG compression artifacts. Skew and rotation levels are still minor though. Ground truth data exists for every file, either self created from OCR with postcorrection or from available online sources. Images, ground truth data and the accuracy test results can be found in detail in the respective GitLab repository.[67]

66 *Smith, Ray,* Building a Multilingual OCR Engine: Training LSTM Networks on 100 Languages and Test Results (Google Inc., June 20, 2016), 16, 17, https://github.com/tesseract-ocr/docs/blob/master/das_tutorial2016/7Building%20a%20Multi-Lingual%20OCR%20Engine.pdf.

67 https://gitlab.ub.uni-bielefeld.de/sfb1288inf/ground_truth_test.

Table 1: Accuracy test results (self-created)

	Quality	Quality features	Document	Pages	CER[a]	WER[b]	CER[c]	WER[d]
Fraktur	High	300 DPI, self created TIFF scans	Estor – Rechts-gelehrsamkeit	4	19.45	45.58	23.61	58.78
			Luz – Blitz	4	19.91	44.54	22.28	61.76
	Middle	TIFFS created from PDFs	Die Gegenwart	10	12.07	19.71	5.38	10.42
English	High	300 DPI, self created TIFF scans	Inside Germany	10	2.60	5.86	1.62	1.88
	Middle	TIFFS created from PDFs	Germans Past and Present	10	4.17	5.69	4.00	5.13

a Additional binarization with Ocropus.
b Additional binarization with Ocropus.
c Only internal Tesseract binarization.
d Only internal Tesseract binarization.

We tested the OCR pipeline with every input document twice, each time using different parameters. The first run utilized the additional binarization step using ocropus-nlbin from ocropy. After the OCR process had finished, we calculated CER and WER with the ocrevalUAtion tool. For the second test run we omitted the additional binarization step and only used the internal binarization step provided by Tesseract. CER and WER were calculated accordingly.

As we can see the pipeline achieves low error rates for Modern English high quality input images. CER is 2.6 and WER is 5.86. When not using the additional binarization step the results are even better with CER being 1.62 and WER being 1.88. If we compare those findings to the results published by Ray Smith, we can see that we achieved slightly better results. We attribute this to the high quality of our self created input images. They were created in accordance with our own best practices, as outlined above.

Error rates for English medium quality input images are slightly worse but still in close range to the error rates of high quality input images.

In general, the error rates for Fraktur text are much higher than for English text. This could be linked to several factors. First, as mentioned above, the model for German Fraktur is not available from the tessdata_best set. Second, because of their age, Fraktur texts are more often susceptible to background noise, touching or broken characters and geometric formation. This could also explain the higher error rates for supposedly high quality Fraktur input images compared to medium quality input images. On paper the high quality input images have a much higher DPI but suffer more from bleed through, line skew and broken characters (fading characters), etc. than the middle quality input images.

From these findings we can conclude that the quality level of the input images should be seen as a two dimensional parameter consisting of technical quality (DPI, lossless compression, etc.) and physical quality of the text (fading characters, skew, bleed through, etc.).

Regarding the additional binarization step, it is hard to judge when it is beneficial. For the OCR of Fraktur text the results suggest that it could be beneficial in some cases. Possible researchers should run the OCR process twice, once with additional binarization and once without it, and judge for themselves which output has fewer errors.

For English text the results suggest that additional binarization is not beneficial.

To give a finite answer we would have to do more testing with more diverse ground truth data.

Natural language processing

By using natural language processing (NLP) methods it is possible to enrich plain texts with various useful information. Our goal after processing is to make the source searchable for the added data. For that purpose, we decided to use the free open source NLP library spaCy. It is fast, reliable and offers natural language processing for all languages used in our context to the same extent. The latter is not self-evident, other open source approaches we have tried, like *Stanford CoreNLP*, do not provide all features we want to make use of for all languages.

It was important that we were able to handle each text corpus in the same way, independently of the input language. For further work with the gathered data we use the software collection *The IMS Open Corpus Work-*

bench (CWB) which uses a data type called *verticalized text (vrt)* format. This data type is a fairly uncommon variation of XML, which is why most of the NLP libraries do not offer it as an output option. Because we did not want to perform much data type conversion, we needed a flexible NLP toolkit with which we could configure the output format with an, in the best case, application programming interface (API).

For the time being it is enough for us to use four methods for further text analysis. These are tokenization, lemmatization, part-of-speech tagging (POS tagging) and named entity recognition, which are all described in the following. It is good to know that our chosen natural language processing toolkit offers this and gives us the possibility to extent this portfolio with more features in the future.

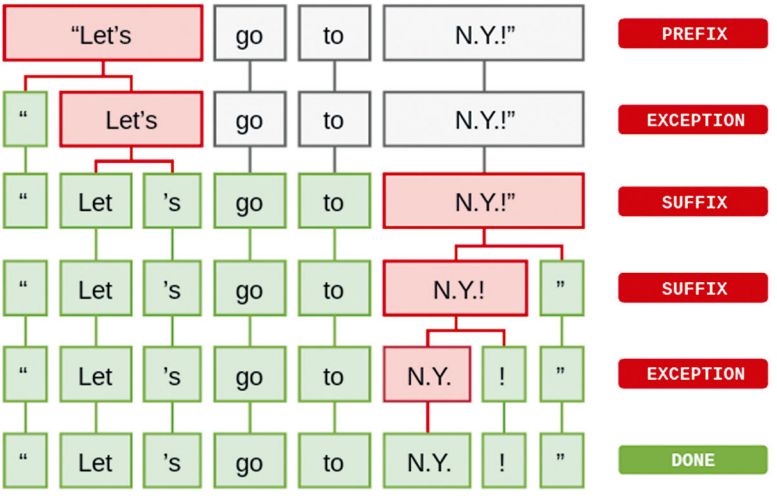
After performing NLP on a text, it is possible to fulfill queries, like "`show all text passages where words of the category adjective appear around the word 'world' in all its reflections within a maximum word spacing of 5 words`", which can help you by finding assessments made of the world in the queried text.

5. Tokenization

Tokenization is the task of splitting up a text into so called tokens. These can either be words or punctuation marks.[68] This process can be easily explained by a short example. The sentence "Let's go to N.Y.!" will get tokenized into eight tokens: ", `Let`, `'s`, `go`, `to`, `N.Y.`, `!`, ". As you can see, tokenization is not just about splitting by white space or non-alphanumeric characters. The tokenizer system needs to recognize that the dots between and after the "N" and "Y" do not indicate the end of a sentence, and that "Let's" should not result in one token but in two: "`Let`" and "'s". The process of tokenizing differs a bit in each software implementation. The spaCy tokenizer, which is used by us, is shown in figure 7.

68 *Manning, Christopher D./Raghavan, Prabhakar/Schutze, Hinrich,* Introduction to Information Retrieval, Cambridge: Cambridge University Press, 2008, 22, https://doi.org/10.1017/cbo9780511809071.

Fig. 7: Tokenization process with spaCy



First, the tokenizer splits the sentence into tokens by white space. After that, an iterative process starts in which the tokenizer loops over the gained tokens until the process is not interrupted by exceptions anymore. An exception occurs when tokenization rules exist that can be applied to a token.[69]

The tokenization rules which trigger the exceptions are usually extremely specific to the language used in the text. SpaCy offers the ability to extend its features with predefined language packages for all languages we want to support. These packages already include the language specifics. They are easily expandable by adding your own rules for special expressions in texts. The latter can be the case if your text uses an old style of a language – which is likely to appear in historical texts – which is not processable by modern tokenization rules that are designed for today's language compositions.

69 Image from *Explosion AI*, Linguistic Features · spaCy Usage Documentation: Tokenization, https://spacy.io/usage/linguistic-features#tokenization [accessed: 25.03.2019].

Lemmatization

The grammar, which forms a language, leads to different forms of a single word. In many situations where you want to use methods for text analysis, you are not interested in a specific form but in the occurrence of any form. To make it possible to query for those occurrences we use a lemmatization process, which adds a lemmatized version to each token of a text. The lemmatized version of a word is defined as the basic form of it, like the one which can be found in a dictionary.[70] As an explanatory example the following words are lemmatized like this:

- `writes`, `wrote`, `written` → `write`
- `am`, `are`, `is` → `be`
- `car`, `cars`, `car's`, `cars'` → `car`

The lemmatization process is based on a dictionary lookup where a lemma dictionary (basically a long list of words in all its inflections with a corresponding lemmatized entry) is used to lemmatize words. These dictionaries are usually bundled in the NLP toolkit's language packages. If the software does not find a word which should be lemmatized in the corresponding dictionary, it just returns the word as it is. This is a pretty naive implementation, which is why for some languages the developers added some generic rules which are applied after a dictionary lookup fails.

For our chosen NLP toolkit, the lemmatization implementations can be found in the source code repositories of Explosion AI. These repositories show why their English lemmatizer performs better than most of their others like, for example, the German one. It is not only based on dictionaries but also on more generic rules specific to the language.[71, 72]

70 *C. D. Manning/P. Raghavan/H. Schutze*, Introduction to Information Retrieval, 32.

71 *Explosion AI*, Industrial-Strength Natural Language Processing (NLP) with Python and Cython: SpaCy/Spacy/Lang/de/Lemmatizer.py, https://github.com/explosion/spaCy/tree/v2.1.0/spacy/lang/de/lemmatizer.py [accessed: 21.05.2019].

72 Ibid.

Part-of-speech tagging

Words can get categorized into different classes. Knowing the class of a word is useful because they offer further information about the word itself and its neighbors. Part-of-speech tagging is exactly about that: Tokenized texts are analyzed, and a category of a predefined set is assigned to each token.[73] These category sets are also called 'tagsets'. SpaCy's POS tagging process is based on a statistical model which improves the accuracy of categorization predictions by using context related information, for example, a word following "the" in English is most likely a noun.[74] The part-of-speech tagsets used by spaCy are based on the chosen language model.[75]

Named entity recognition

Named entity recognition (NER) is the task of detecting named entities in a text. These can be understood as anything that can be referred to with a name, like organizations, persons and locations. Named entities are quite often ambiguous, the token `Washington`, for example, can refer to a person, location, and an organization.[76] Our used NLP software offers a system that detects named entities automatically and assigns a NER-tag to the corresponding token. The release details of spaCy's language packages list all NER-tags that are assigned by it.

Accuracy

Having NLP software that satisfies your needs in terms of functionality is important, nevertheless you should be aware of its reliability. Table 2 gives an overview of the accuracies of spaCy's language packages.

---

73 *Jurafsky, Daniel/Martin, James H.*, Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition, Draft of September 23, 2018, 2018, 151, 156.

74 *Explosion AI*, Linguistic Features spaCy Usage Documentation: Tokenization.

75 *Explosion AI*, Annotation Specifications · spaCy API Documentation: Part-of-Speech Tagging, https://spacy.io/api/annotation#pos-tagging [accessed: 27.03.2019].

76 *D. Jurafsky and J. H. Martin*, Speech and Language Processing, 328–29.

Table 2: Accuracy values of spaCy

	NER	POS tagging	Tokenization
Dutch[a]	100.00	87.02	91.57
English[b]	99.07	86.56	96.92
French[c]	100.00	82.64	94.48
German[d]	95.88	83.10	96.27
Greek[e]	100.00	71.58	94.60
Italian[f]	100.00	86.05	95.77
Portuguese[g]	100.00	88.85	80.36
Spanish[h]	100.00	89.46	96.92

These numbers are gained from tests made by Explosion AI, the organization behind spaCy. They tested their language models with data similar to the data the models are based on, these are mostly Wikipedia text sources. This means that these numbers can not be assigned to all text genres we are processing. Until now we have not made accuracy tests with our data but we expect lower accuracies.

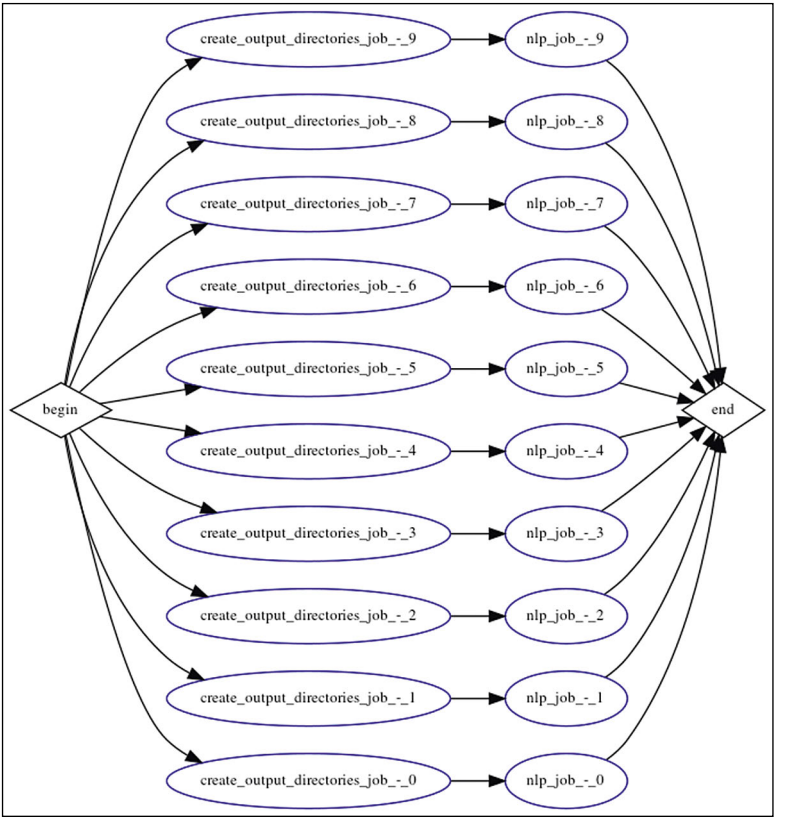
a *Explosion AI*, Nl_core_news_sm, version 2.1.0 (2019), https://github.com/explosion/spacy-models/releases/tag/nl_core_news_sm-2.1.0.
b *Explosion AI*, En_core_web_sm, version 2.1.0 (2019), https://github.com/explosion/spacy-models/releases/tag/en_core_web_sm-2.1.0.
c *Explosion AI*, Fr_core_news_sm, version 2.1.0 (2019), https://github.com/explosion/spacy-models/releases/tag/fr_core_news_sm-2.1.0.
d *Explosion AI*, De_core_news_sm, version 2.1.0 (2019), https://github.com/explosion/spacy-models/releases/tag/de_core_news_sm-2.1.0.
e *Explosion AI*, El_core_news_sm, version 2.1.0 (2019), https://github.com/explosion/spacy-models/releases/tag/el_core_news_sm-2.1.0.
f *Explosion AI*, It_core_news_sm, version 2.1.0 (2019), https://github.com/explosion/spacy-models/releases/tag/it_core_news_sm-2.1.0.
g *Explosion AI*, Pt_core_news_sm, version 2.1.0 (2019), https://github.com/explosion/spacy-models/releases/tag/pt_core_news_sm-2.1.0.
h *Explosion AI*, Es_core_news_sm, version 2.1.0 (2019), https://github.com/explosion/spacy-models/releases/tag/es_core_news_sm-2.1.0.

Implementation and workflow

Like our OCR implementation the NLP process is also implemented as a software pipeline. It is capable of processing text corpora in Dutch, English, French, German, Greek, Italian, Portuguese, and Spanish. It will only accept raw text files as input and provides verticalized text files as a result. The pipe-

line contains only one processing step, which is the spaCy natural language processing. While implementing this as a pipeline may sound laborious, it gives us the flexibility to easily extend the pipeline in the future.

Fig. 8: NLP pipeline procedure of ten input files



In order to achieve good computational performance, we aimed to make use of modern multicore systems. For that purpose, we used pyFlow, a powerful parallel task processing engine. Figure 8 shows one pipeline run where ten input files are processed. Each of these files is treated in a separate task that can run parallel to the others if the hardware is capable of doing parallel computation.

The NLP pipeline is deployed in a *Linux* container; for that purpose, we created a Dockerfile which tells the Docker build system to install all dependencies which are needed for using our NLP pipeline in a container image. The source code of this is available at the GitLab system hosted by Bielefeld University.[77] There you will find instructions on how to build and use the image or, in case you do not want to build the image yourself, we also offer an image in a prebuilt state.[78] The published image contains spaCy[79] in version 2.1.0 and language packages for processing Dutch, English, French, German, Greek, Italian, Portuguese and Spanish texts. All Software that is needed to realize and use this image is completely free and open source.

After the image was created, we were able to start multiple instances of the natural language processing software, encapsulated in Linux containers. Each instance executes one NLP pipeline run which processes the input data. An execution is bound to one specific text language, so the files processed within one pipeline run must contain texts in the same language.

Our usual workflow is described in the following:

1. Receive text corpora as raw text files
2. Create input and output directories
3. Copy files into the input directory
4. Start the NLP software
   - `nlp -i <inputdir> -l <languagecode> -o <outputdir>`
5. Check the results in the output directory

The results are saved as verticalized text files. This is a XML compliant format, where each line contains one token with all its assigned attributes. One line is structured in the following order: `word, lemmatized word, simplified part-of-speech tag, part-of-speech tag, named entity recognition tag`. NULL indicates that no named entity is recognized. The beginning and end of a sentence is represented by <s> and </s>

77 *Jentsch, Patrick/Porada, Stephan*, Natural Language Processing, version 1.0, 2019, https://gitlab.ub.uni-bielefeld.de/sfb1288inf/nlp/tree/from_text_to_data [accessed: 31.08.2019].

78 *P. Jentsch/S. Porada*, Docker Image: Natural Language Processing.

79 *Explosion AI*, SpaCy, version 2.1.0 (2019), https://github.com/explosion/spaCy/releases/tag/v2.1.0 [accessed: 31.08.2019].

and analog to this you have a start and an end tag for the text and the complete corpus.

As a short example, we process the text "Tesseract is a software maintained by Google." The resulting verticalized text file looks as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<corpus>
<text>
<s>
   Tesseract tesseract NOUN NN NULL
   is be VERB VBZ NULL
   a a DET DT NULL
   software software NOUN NN NULL
   maintained maintain VERB VBN NULL
   by by ADP IN NULL
   Google google PROPN NNP ORG
   . . PUNCT . NULL
</s>
</text>
</corpus>
```

spaCy and Stanford CoreNLP

Before we decided to use spaCy as our natural language processing software we worked with Stanford CoreNLP. This software is versatile but does not offer its full functionality for all languages used in our context. In order to work with the processed texts, we decided to use a software called The IMS Open Corpus Workbench (CWB), which requires the verticalized text file format as input. With Stanford CoreNLP we had to write complex conversion programs to transfer the output into the desired verticalized text format in order to be able to import it to our CWB instance.

Table 3 shows our functional requirements and those supported by *Stanford CoreNLP* for specific languages.[80]

80 *Stanford University*, Using Stanford Corenlp on Other Human Languages, https://stanfordnlp.github.io/CoreNLP/human-languages.html#models-for-other-languages [accessed: 27.03.2019].

Table 3: Functionality of Stanford CoreNLP

	English	French	German	Spanish
Tokenization	x	x		x
Lemmatization	x			
Part-of-Speech Tagging	x	x	x	x
Named Entity Recognition	x		x	x

To handle the mentioned problems we switched to spaCy, which offers the needed functionalities for all languages we encounter and also has a good application programming interface, which we were able to utilize in order to create the needed verticalized text files.

6. Conclusion

With our pipeline we hope to encourage scientific researchers, not only at Bielefeld University, to make use of the means provided by the digital age. Since the cooperation between information science and the humanities lies at the very core of Digital Humanities, we want to contribute our share by providing a tool to digitize and process texts. This tool is constantly revised and improved in accordance with the needs of its potential users and in close collaboration with them.

Bibliography

Adobe Systems Incorporated, Document Management – Portable Document Format – Part 1: PDF 1.7 (2008), 25 ff., https://www.adobe.com/content/dam/acom/en/devnet/pdf/PDF32000_2008.pdf [accessed: 26.03.2019].

Adobe Systems Incorporated, TIFF: Revision 6.0, version 6.0 (1992), 57–58, https://www.adobe.io/content/dam/udp/en/open/standards/tiff/TIFF6.pdf [accessed: 31.08.2019].

Berlin-Brandenburgische Akademie der Wissenschaften, Ziel und Fokus des DTA-Basisformats (Deutsches Textarchiv, Zentrum Sprache der Berlin-Brandenburgischen Akademie der Wissenschaften), http://www.deutsches

textarchiv.de/doku/basisformat/ziel.html#topic_ntb_5sd_qs__rec [accessed: 12.03.2019].

Boettiger, Carl, An Introduction to Docker for Reproducible Research, in: ACM SIGOPS Operating Systems Review 49 (2015), 71–79, https://doi.org/10.1145/2723872.2723882.

Carrasco, Rafael C., Text Digitisation, https://sites.google.com/site/textdigitisation/ [accessed: 11.06.2019]

Chaudhuri, Arindam et al., Optical Character Recognition Systems for Different Languages with Soft Computing, Springer International Publishing: 2017, 90–92, 17–22, https://doi.org/10.1007/978-3-319-50252-6.

Cheriet, Mohamed, et al., Character Recognition Systems, John Wiley & Sons, Inc.: 2007, 8–15, https://doi.org/10.1002/9780470176535.

Debian, version 9 (The Debian Project, 2017), https://www.debian.org/ [accessed: 31.08.2019].

Docker, version 18.09.1 (Docker, 2013), https://www.docker.com/ [accessed: 31.08.2019].

Explosion AI, Linguistic Features · spaCy Usage Documentation: Tokenization, https://spacy.io/usage/linguistic-features#tokenization [accessed: 25.03.2019].

Explosion AI, Industrial-Strength Natural Language Processing (NLP) with Python and Cython: SpaCy/Spacy/Lang/de/Lemmatizer.py, https://github.com/explosion/spaCy/tree/v2.1.0/spacy/lang/de/lemmatizer.py [accessed: 21.05.2019].

Explosion AI, Annotation Specifications · spaCy API Documentation: Part-of-Speech Tagging, https://spacy.io/api/annotation#pos-tagging [accessed: 27.03.2019].

Explosion AI, SpaCy, version 2.1.0 (2019), https://github.com/explosion/spaCy/releases/tag/v2.1.0 [accessed: 31.08.2019].

Google Inc., Tesseract OCR (2019), https://github.com/tesseract-ocr/tesseract/ [accessed: 31.08.2019].

Google Inc., ImproveQuality: Improving the Quality of the Output (2019), https://github.com/tesseract-ocr/tesseract/wiki/ImproveQuality [accessed: 31.08.2019].

Hardie, Andrew/Evert, Stefan, IMS Open Corpus Workbench, http://cwb.sourceforge.net/ [accessed: 31.08.2019].

International Organization for Standardization, ISO/Iec 9126-1:2001: Software Engineering – Product Quality – Part 1: Quality Model (International

Organization for Standardization, June 2001), https://www.iso.org/standard/22749.html [accessed: 31.08.2019].

International Organization for Standardization, ISO/Iec 25010:2011: Systems and Software Engineering – Systems and Software Quality Requirements and Evaluation (SQuaRE) – System and Software Quality Models (International Organization for Standardization, March 2011), https://www.iso.org/standard/35733.html [accessed: 31.08.2019].

Jackson, Mike/Crouch, Steve/Baxter, Rob, Software Evaluation: Criteria-Based Assessment (Software Sustainability Institute, November 2011), https://software.ac.uk/sites/default/files/SSI-SoftwareEvaluationCriteria.pdf [accessed: 31.08.2019].

Jentsch, Patrick/Porada, Stephan, Docker Image: Optical Character Recognition, version 1.0, Bielefeld 2019, https://gitlab.ub.uni-bielefeld.de/sfb1288inf/ocr/container_registry [accessed: 31.08.2019].

Jentsch, Patrick/Porada, Stephan, Natural Language Processing, version 1.0, 2019, https://gitlab.ub.uni-bielefeld.de/sfb1288inf/nlp/tree/from_text_to_data [accessed: 31.08.2019].

Jurafsky, Daniel/Martin, James H., Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition, Draft of September 23, 2018, 2018, 151, 156.

Manning, Christopher D./Raghavan, Prabhakar/Schutze, Hinrich, Introduction to Information Retrieval, Cambridge: Cambridge University Press, 2008, 22, https://doi.org/10.1017/cbo9780511809071.

Olah, Christopher, "Understanding LSTM Networks, https://colah.github.io/posts/2015-08-Understanding-LSTMs/ [accessed: 26.03.2019].

Oztan, Bazak, et al., Removal of Artifacts from JPEG Compressed Document Images, in: Reiner Eschbach/Gabriel G. Marcu(eds.), Color Imaging XII: Processing, Hardcopy, and Applications, (SPIE) 2007, 1-3, https://doi.org/10.1117/12.705414.

Red Hat Inc., What's a Linux Container?, https://www.redhat.com/en/topics/containers/whats-a-linux-container [accessed: 20.05.2019].

Saunders, Chris, PyFlow: A Lightweight Parallel Task Engine, version 1.1.20, 2018, https://github.com/Illumina/pyflow/releases/tag/v1.1.20 [accessed: 31.08.2019].

Smith, Ray, Building a Multilingual OCR Engine: Training LSTM Networks on 100 Languages and Test Results (Google Inc., June 20, 2016), 16, 17, https://

github.com/tesseract-ocr/docs/blob/master/das_tutorial2016/7Building%20a%20Multi-Lingual%20OCR%20Engine.pdf.

Smith, Ray, An Overview of the Tesseract OCR Engine, in: Ninth International Conference on Document Analysis and Recognition (ICDAR 2007) Vol 2 (2007), https://doi.org/10.1109/icdar.2007.4376991.

Smith, Ray, A Simple and Efficient Skew Detection Algorithm via Text Row Accumulation, in: Proceedings of 3rd International Conference on Document Analysis and Recognition (IEEE Comput. Soc. Press, 1995), https://doi.org/10.1109/icdar.1995.602124.

Sourceforge, CWB/Perl & Other APIs, http://cwb.sourceforge.net/doc_perl.php [accessed: 13.05.2019].

Stanford University, Using Stanford Corenlp on Other Human Languages, https://stanfordnlp.github.io/CoreNLP/human-languages.html#models-for-other-languages [accessed: 27.03.2019].

Text Encoding Initiative, TEI P5: Guidelines for Electronic Text Encoding and Interchange (TEI Consortium) https://tei-c.org/release/doc/tei-p5-doc/en/Guidelines.pdf [11.06.2019].

University of Alicante, ocrevalUAtion, https://github.com/impactcentre/ocrevalUAtion [accessed: 05.04.2019].

University of Alicante, OcrevalUAtion, version 1.3.4 (2018), https://bintray.com/impactocr/maven/ocrevalUAtion [accessed: 31.08.2019].

Ye, Peng/Doermann, David, Document Image Quality Assessment: A Brief Survey, in: 2013 12th International Conference on Document Analysis and Recognition (2013), 723, https://doi.org/10.1109/icdar.2013.148.